

ACCESSING THE TRS-80™ ROOM



VOLUME
ONE

ILLUSTRATED
BY
1980
INFINITY
GRAPHICS

THE BØØK

ACCESSING THE TRS-80 ROM

VOLUME I: MATH

Raymond E. Daly IV
Stephen C. Hill
Roy Soltoff
Thomas B. Stibolt, Jr.
Richard P. Wilkes

ILLUSTRATIONS BY
INFINITY GRAPHIX

Insiders Software Consultants
P.O. Box 2441,
Springfield, VA 22152

Copyright (c) 1980 by Insiders Software Consultants, Inc.
All rights reserved.

First Edition 1980

Reproduction in any manner, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without expressed written permission, is prohibited.

Disclaimer:

While Insiders Software Consultants, Inc. has taken every precaution in the preparation of this book, it assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

Radio Shack and TRS-80 are registered trademarks of the Tandy Corporation.

Insiders Software Consultants, Inc.
PO Box 2441,
Springfield, VA 22152

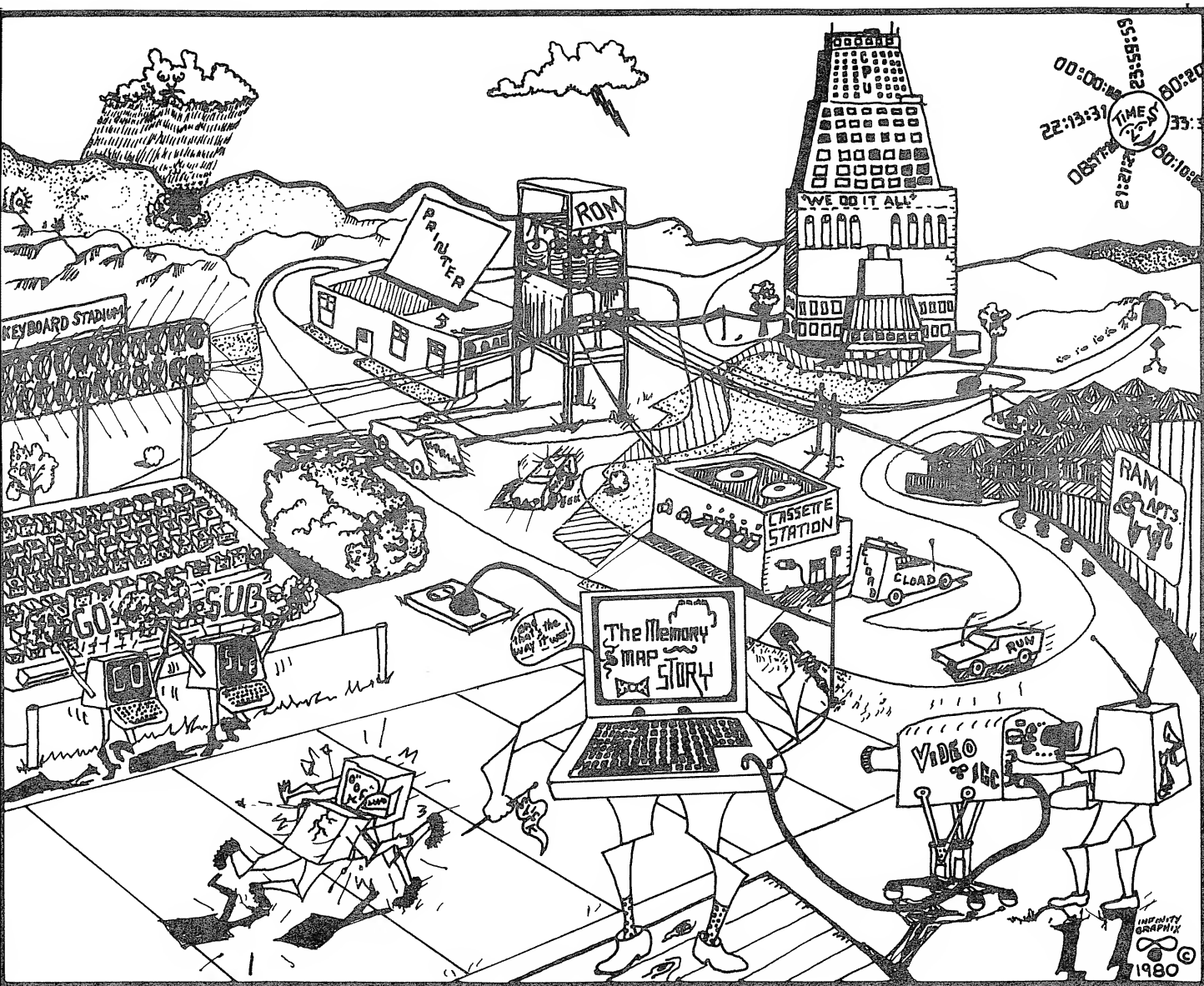


TABLE OF CONTENTS

Preface	0000 0111
Introduction	0000 1000
Chapter 1: Formats and Accumulators	
BASIC Data Areas	1-1
Integer Data Format	1-2
Single Precision Data Format	1-3
Double Precision Data Format	1-6
Memory Accumulators	1-8
Register Accumulators	1-9
Chapter 2: Data Manipulation	
Variable Type Flag	2-1
Error Recovery	2-2
Data Conversion	2-3
Moving Data	2-5
ASCII to Binary Conversion	2-7
Binary to ASCII Conversion	2-8
Formatted Conversion From Binary to ASCII	2-9
Chapter 3: Assembly Interfacing	
Register Utilization	3-1
Integer Math Routines	3-1
Single Precision Math Routines	3-3
Double Precision Math Routines	3-6
Interfacing to Math Functions	3-7
ABS(X) - Absolute Value Function	3-8
ATN(X) - Arctangent Function	3-8
COS(X) - Cosine Function	3-8
EXP(X) - Exponential Function	3-9
FIX(X) - Truncation Function	3-9
INT(X) - Greatest Integer Function	3-10
LOG(X) - Natural Logarithm Function	3-10
POWER - Raise X to the Power of Y	3-11
RND(X) - Random Number Generation Function	3-12
SGN(X) - Determine Sign of Argument	3-13
SIN(X) - Sine Function	3-14
SQR(X) - Square Root Function	3-14
TAN(X) - Tangent Function	3-14

Chapter 4: Disassembly of ROM Math Routines	
Single Precision Addition and Subtraction	4-2
Log Routine Data Values	4-6
Log Routine (LOG)	4-6
Single Precision Multiplication	4-7
Single Precision Division	4-8
Absolute Value Routine (ABS)	4-12
Sign Determination Routine (SGN)	4-12
Single Precision Comparison	4-15
Integer Comparison	4-15
Double Precision Comparison	4-16
Convert to Integer (CINT)	4-16
Convert to Single Precision (CSNG)	4-17
Convert to Double Precision (CDBL)	4-17
FIX Routine	4-18
INT Routine	4-19
Integer Subtraction	4-21
Integer Addition	4-21
Integer Multiplication	4-21
Absolute Value of an Integer	4-23
Double Precision Subtraction	4-23
Double Precision Addition	4-23
Double Precision Multiplication	4-28
Double Precision Division	4-29
Convert Hex Data to ASCII	4-45
Convert Integer to ASCII	4-46
Square Root (SQR)	4-49
Raising to a Power	4-49
Exponentiation (EXP)	4-50
Random Function (RND)	4-52
Cosine Function (COS)	4-54
Sine Function (SIN)	4-54
Tangent Function (TAN)	4-55
Arctangent Function (ATN)	4-55
Appendix A: Label Table	A-1
Appendix B: Interfacing Examples	
Convert Double Precision to Hex	B-1
Solve Quadratic Equation	B-4
Random Number Generator Test	B-8
Appendix C: BASIC Disassembler Listing	C-1

PREFACE

This book has been written in a manner that resists non-sequential access methods. In other words, it is HIGHLY recommended that the reader start with the Introduction and continue on through the book. After the first reading, the Table of Contents can double as an Index, which was left out to discourage people from leaping into strange waters.

INTRODUCTION

Since you are reading this book, you are most likely the owner of a TRS-80 microcomputer. In spite of the disdain shown for your beast by those who don't know any better, you have probably found that it is generally adequate for most of your personal computing needs. Unfortunately, due to either disinterest or lack of knowledge, various organizations which should be making essential information available to you are failing to do so. This book is a response to the obvious desire of you--the owner, operator, and programmer of a TRS-80--to know as much about this machine as possible. It is our intention to make available to you information not readily obtainable from any other source.

This is the first of a three volume set that will detail the operation of the Level II ROM. Explicit descriptions will be given so that ROM routines, accessible from assembly language, may be used. The series will explore Level II BASIC at a depth that will be satisfactory for even the most skilled programmer and yet will still prove to be useful to the novice programmer with only minimal machine language experience.

This first volume describes in detail how numerical data is stored and manipulated in memory. In addition, it provides the complete assembly language interfacing procedures for all mathematical functions including addition, subtraction, multiplication, and division of integer, single precision, and double precision values. Also, the code necessary for accessing the logarithmic, trigonometric, and comparison routines is provided along with examples on how to manipulate data in memory. Although Input/Output will be covered in Volume II, complete instructions are provided for inputting and outputting numerical data. In Appendix A, the user will find a complete list of entry points and data areas used by Level II, including some areas used by DOS and Disk BASIC. This list provides a quick reference to all routines discussed in this volume and in the ones to follow.

The wealth of information that one will find in this book is substantial. Due to the amount of material available, you will find the next two volumes equally as informative. Topics will include data and tape formats, I/O routines, BASIC program storage, the editor, the parser, error-handler, and much, much more. Please refer to the coupon in the back of this book for a discount on the next volume in the series.

We welcome any comments or suggestions. Please feel free to write us.

Thank you and good luck,

Insiders Software Consultants, Inc.
PO Box 2441,
Springfield, VA 22152

CHAPTER 1: FORMATS AND ACCUMULATORS

In order to fully understand the operation of the mathematics routines, one must have a basic understanding of the internal format of numerical data. The TRS-80 supports three types of numerical variables: integer, single precision (also known as "floating-point"), and double precision. In order to manipulate the different values, Level II utilizes various memory accumulators in low RAM memory. In the following sections, the format of these three types of values will be discussed along with the format of the memory accumulators.

Data in the form of integers, single precision, and double precision real numbers are stored in RAM memory. This data is typically stored in locations which are associated with variable names that are used in a BASIC program. Level II creates and maintains two tables of variable names and their associated values. The two tables of variables immediately follow the end of the BASIC program in memory. It is for this reason that whenever the program is modified (after a BREAK for example), all variable values are cleared and the pointers are reset.

The first variable area is for simple (i.e., non-array) variables. Variables for each of the four data types (strings included) can be stored in this region. The beginning of the simple variable storage area is indicated by a two-byte pointer, SCLERS, that is stored at locations 40F9H and 40FAH. Simple variables are placed in this region as they are encountered during program execution. Therefore, since the search through this table is done sequentially, all frequently used variables should be initialized at the beginning of the program before any program statements but following the CLEAR and DEF commands. This step is very important when one uses long programs with many variables. Early definition of variables used in FOR/NEXT loops first is very worthwhile.

The entry for each different type of variable follows a set format. The first byte of each entry is for the type of variable; two (2) for integer, three (3) for string, four (4) for single precision, and eight (8) for double precision. The next two bytes consist of the two significant characters in the variable name stored in reverse order. Thus, the variable name "AB" would be stored in the RAM variable table as "BA." However, in the case of

single character variable names, the first byte will be zero (00) instead of being stored as a space. Numeric data follows the name, while string data is pointed to by an address following the name. The code representing the variable type also reflects the length of the following data. Accordingly, each integer will have two bytes, strings-three, single precision-four, and double precision will have eight.

The two bytes of the integer are stored, as is standard, least significant byte (LSB) first, followed by the most significant byte (MSB). Integers in the TRS-80 are signed, meaning that they are either positive or negative. The system used to represent integers is called "two's complement." In this representation, the most significant bit of the sixteen bits used for integer storage is the sign bit. The sign bit is set (1) for negative numbers and is zero (0) for positive values. Positive values are stored in the low order fifteen bits as a standard binary value. Therefore, a value of +3 would be represented as 00000000 00000101.

Negative numbers, on the other hand, are represented as if the corresponding positive value including the sign (zero for positive) had been complemented and then one was added to the result. Complementing is the process of changing each of the sixteen bits into its opposite; zero to one, one to zero. If the above process is performed twice on any sixteen bit value, the original number will always be returned. Any negative value number can be decoded by taking its two's complement and placing a minus sign (-) in front of it. Thus, the value 1234 in decimal will be represented as the sixteen bit value 00000100 11010010 (04D2H) and the value -1234 as 11111011 00101110 (FB2EH).

As stated previously, integer values are stored in memory in two contiguous bytes. The least significant byte (rightmost in the above examples) is stored in the lowest memory location and the most significant byte in the next higher memory location. Thus, the value for 1234 which in hex is 04D2H would be stored D2H 04H.

Using two's complement, the largest integer value that can be stored in memory is 32,767 (7FFFH) or, in binary, 01111111 11111111. The smallest negative value that can be represented using integers is -32,768 (8000H) or, in binary, 10000000 00000000. Values that do not fall within the -32,768 to +32,767 range must be represented as either single precision or double precision numbers.

Single precision numbers are handled in a totally different manner. Most readers should be familiar with scientific notation, a technique for representing a number as a real number between one and ten (called the mantissa) and an integer that represents the power of ten the real number must be multiplied by to produce the original value. As an example, the number 378.662 would be represented as $3.78662 \times 10^{**2}$ (the two asterisks (**) meaning "raised to the power of"). In the TRS-80 and most computers, this value would be printed as 3.78662E+02, which will be the notation followed throughout the rest of this text.

A modification of this process requires that the mantissa always be less than one but greater than or equal to one-tenth ($1 > X \Rightarrow .1$). Using this procedure called normalization, the above example would be represented as .378662E+03.

Similarly, values can be represented with a binary mantissa and exponent (which now represents a power of two by which the binary mantissa must be multiplied in order to produce the original value). In this case, bits to the right of the binary point represent increasingly negative powers of two. Thus, $1/8$ would be represented as .001 [$1/(2^{**3})$]. Examples are:

$$\begin{aligned} .5_{10} &= .1_2 \quad (2^{-1}) \\ .25_{10} &= .01_2 \quad (2^{-2}) \\ .125_{10} &= .001_2 \quad (2^{-3}) \\ .0625_{10} &= .0001_2 \quad (2^{-4}) \end{aligned}$$

The representation of floating point numbers in the TRS-80 as well as other computers uses this concept.

In the TRS-80, four bytes are used to represent single precision (6 significant digits) and eight bytes for double precision (16 significant digits). In both cases, one byte is used to hold the exponent and the remaining three or seven (depending on the type of value) is used to hold the mantissa.

The exponent is stored in "excess 128" notation. This means that an exponent of zero ($2^0 = 1$) is represented by 128 (80H), positive exponents are denoted by values of greater than 128, and negative exponents by values of less than 128. Thus, by subtracting 128 from the value, the true exponent is obtained.

The mantissa of a floating-point number is always normalized, which when working in base 10 is within the range $10^0 > X \geq 10^{-1}$. However, the TRS-80 stores the values in BINARY normalized form which means the values lie within the range $[2^0 > X \geq 2^{-1}]$. A little simple arithmetic will show that this range, base 10, is $[1 > X \geq .5]$. All this means is that there will always be a one (1) immediately to the right of the decimal point when the binary normalized mantissa is shown in its binary form.

As an example, the number 72.0 decimal when converted to hex (base 16) is 48H. This value in binary is 01001000. Now, let's normalize this binary number, how about 0.1001000? We have effectively moved the binary point seven places to the left, or divided it by 2^7 . Therefore, to produce the original value from 0.1001000, we must multiply this value by 2^7 . Hence, the representation for 72 decimal in binary normalized form would be:

$$0.1001000 \times 2^7$$

Now that we know what the external representation of this value is, we must now convert it to the TRS-80's internal representation.

The first point here is that, as stated before, the exponent is stored in "excess 128 (80H)" notation. Therefore, our positive exponent of seven would be converted to 87H by adding 80H (effectively setting the high order bit). Now that the exponent problem has been solved, we will move to the mantissa....

Here, one may get a bit confused. First of all, the mantissa will be stored as three bytes for single precision and seven bytes for double precision. Here is how our normalized mantissa would be represented using the full three bytes of a single precision value:

```
.10010000 00000000 00000000
    90H      00H      00H
```

That's fairly obvious; however, this does not provide for a sign bit, which of course is needed. The solution is a very good one. First of all, as you recall, the bit immediately to the right of the binary point is always one. Therefore, there is no need to maintain that bit in memory. The bit can then be used as a sign bit, indicating a negative value when set (1) and a positive value when reset (0). Using this procedure, we maintain our original 24 data bits, while also providing for the sign! Thus the MSB would be changed from 10010000 to 00010000 when then bit 7 is used as the sign bit.

One more detail and we will be able to correctly show the representation of the decimal number 72.0 as it is kept in the TRS-80. One must note that the four bytes of the single precision value are stored:

LSB LSB MSB EXP

Thus, the three mantissa bytes must be placed in reverse order, from the least significant to the most significant.

LSB	LSB	MSB	
00000000	00000000	00010000	(note high order
00H	00H	10H	bit of 3rd byte
			is sign bit, de-
			noting a positive
			value)

Now we have all the information. Here is the number as it would appear in memory:

LSB	LSB	MSB	EXP
00000000	00000000	00010000	10000111
00H	00H	10H	87H

Before we continue, a few more examples might prove useful to firm up our understanding of the way in which the TRS-80 handles single and double precision values.

First, we shall examine the number 73.75. This value when converted to binary would be:

$$73.75 = 01001001.11000000$$

Then, normalize this value.

$$.100100111000000 \times 2^7$$

Convert this value to hex, making the high order bit into a sign bit:

MSB	LSB	LSB	EXP
00010011	10000000	00000000	10000111
13H	80H	00H	87H

The last thing we must do is reverse the order of the bytes of the mantissa and we are done:

LSB	LSB	MSB	EXP
00H	80H	13H	87H

One more value, this one negative, should be done. For ease of explanation, let's use -73.75. In binary, we have:

$$-73.75 = -01001001.11000000$$

Placing this value, normalized in the same manner as is +73.75, into four bytes with the high order bit as the sign bit, one gets:

MSB	LSB	LSB	EXP
10010011	10000000	00000000	10000111
93H	80H	00H	87H

Reversing the order, one gets:

LSB	LSB	MSB	EXP
00H	80H	93H	87H

This corresponds the the format of this value of -73.75 in the TRS-80 tables.

Now that single precision values have been discussed fully, it is almost trivial to explain double precision representation. In fact, the only difference is that double precision values use seven bytes of mantissa instead of three, thus utilizing 56 data bits instead of the 24 used in single precision representation. Hence, double precision values are stored:

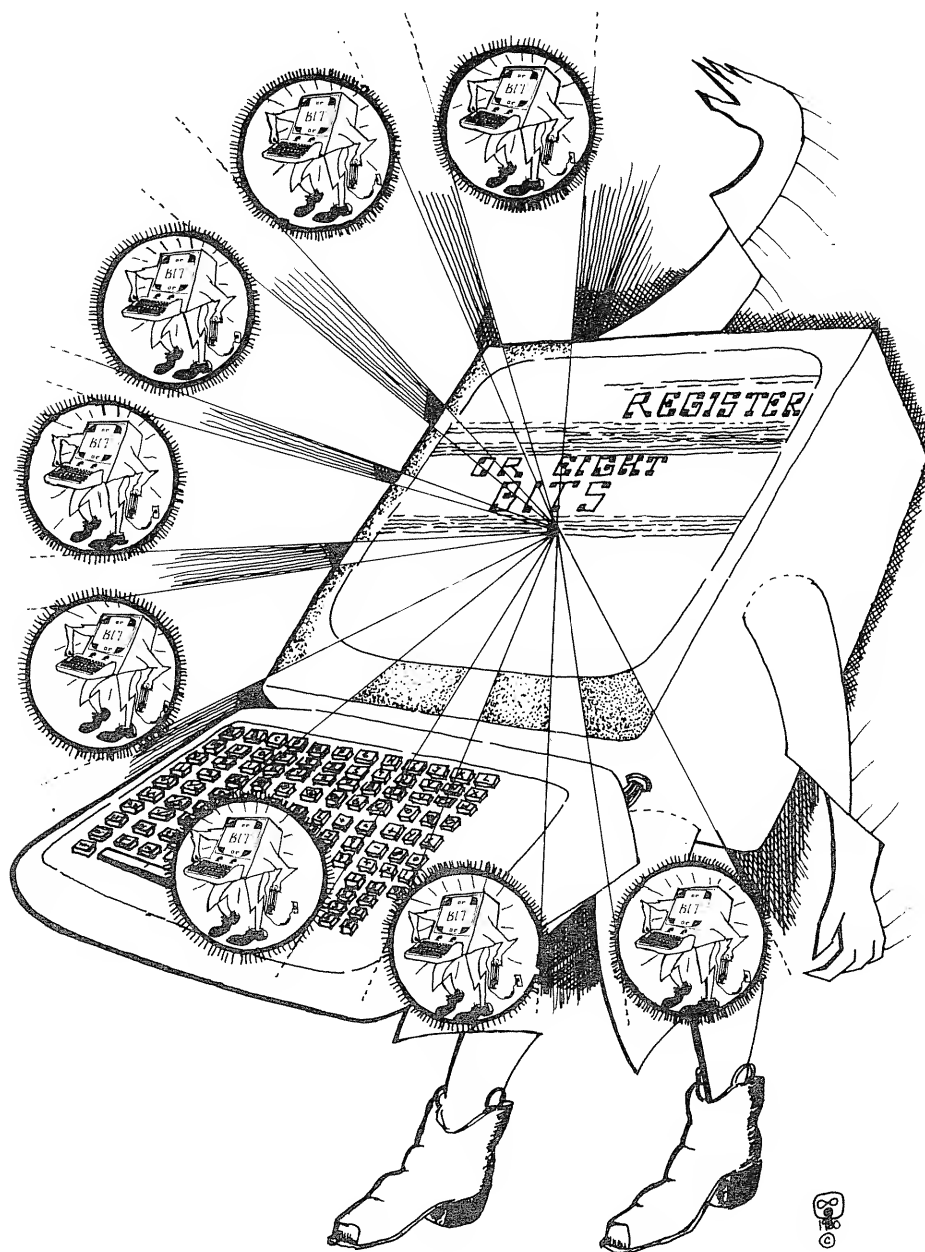
LSB LSB LSB LSB LSB LSB MSB EXP

As an example, we will use 73.75 again. Its double precision equivalent would be:

MSB	LSB	LSB	LSB	LSB	LSB	LSB	EXP
13H	80H	00H	00H	00H	00H	00H	87H

Stored in memory, it would be represented as:

LSB	LSB	LSB	LSB	LSB	LSB	MSB	EXP
00H	00H	00H	00H	00H	80H	13H	87H



MEMORY ACCUMULATORS

To manipulate the various types of data, TRS-80 BASIC utilizes several memory accumulators. In fact, almost all of the the mathematical functions utilize one or more of these areas. In the next sections of this volume, it is assumed that the terms that will be described in the next few pages will be fully understood by the user.

The accumulators that are used for mathematical manipulation are located in low RAM. The areas consist of eight bytes, although depending on the type of variable, not all of the eight bytes need to be utilized.

Each accumulator is organized in much the same format as an area used for the storage of a double precision variable. Let's take the most often utilized accumulator and use it as a model.

All of the memory accumulators are referred to as Floating Point Accumulators (FPA's), even though they are also used in conjunction with integer math. The first accumulator is named FPA1. It resides from 411DH to 4122H. Associated with this accumulator is a type flag (TYPFLG at 40AFH), which stores the type of variable currently present in the accumulator (remember, 8=double, 4=single, 2=integer). The format of this accumulator is as follows:

411DH	411EH	411FH	4120H	4121H	4122H	4123H	4124H
LSB	LSB	LSB	LSB	LSB	LSB	MSB	EXP
(DBL Only)	(DBL Only)	(DBL Only)	(DBL Only)	(SNG DBL)	(SNG DBL)	(SNG DBL)	

The above format holds true for single and double precision values. However, integers are in this format:

4121H	4122H
LSB	MSB

The next floating point accumulator is called FPA2. It is located at addresses 4127H - 412EH. Its type flag (TYPFL2) is stored at 40B0H. The manner in which the values are stored is as follows:

4127H	4128H	4129H	412AH	412BH	412CH	412DH	412EH
LSB	LSB	LSB	LSB	LSB	LSB	MSB	EXP
(DBL Only)	(DBL Only)	(DBL Only)	(DBL Only)	(SNG DBL)	(SNG DBL)	(SNG DBL)	

The last memory accumulator (FPA3) is located at addresses 414AH - 4151H. This accumulator is used by the single precision multiplication and double precision division routines. There is no need for direct user manipulation of this area, but the assembly language programmer should be aware of its use.

Remember that the majority of the math routines must be informed as to the type of data being manipulated. This can be accomplished by setting the type flags directly or through the use of the ROM calls that will be described in the next chapter. For further information on the type flag one can refer to:

TRS-80 Level II User's Manual, Ch. 8, pp. 8-9.

Another type of accumulator utilized by the Level II ROM is a Register Accumulator. Various routines, when dealing with single precision data values, use CPU registers B, C, D, and E to hold the 4-byte value and operate on it. The term "Register Floating Point Accumulator (RFPA)" will be used to refer to these registers collectively. They are utilized as follows:

Register E:	Contains the Least Significant Byte of the mantissa of the Single Precision value.
Register D:	Contains the next, more significant byte of the mantissa.
Register C:	Contains the Most Significant Byte of the mantissa.
Register B:	Contains the Exponent of the Single Precision Value to be manipulated.

Since the registers, when listed in the same order of significance as the value would be represented in memory, would be listed "EDCB," the RFPA may be referenced at times by this mnemonic.

CHAPTER 2: DATA MANIPULATION

As stated earlier, the majority of the math routines must be informed as to the type of data value being processed. Since the Floating Point Accumulators are used to hold integer, single precision, and double precision values, some method must be employed to specify the type currently contained in FPA1, the main accumulator. The codes for each type are as follows:

- 2 - Integer variable
- 3 - String variable
- 4 - Single Precision variable
- 8 - Double Precision variable

One of the above values must be stored in TYPFLG, the type flag for FPA1 located at 40AFH. The TYPFLG can be set to any non-string type through the use of one of the following calls:

```
SETINT  CALL    0A9DH    ;Set TYPFLG for Integer
SETSNG  CALL    0AEFH    ;Set TYPFLG for Single Prec.
SETDBL  CALL    0AECH    ;Set TYPFLG for Double Prec.
```

The type of the contents of FPA1 can easily be determined through the use of the routine at address 25D9H. The routine sets various flags (conditions the flag register, F) depending on the value of TYPFLG. This routine is the one that is ultimately executed through the use of the RST 32, which saves two bytes in a program. The following code sequence will determine the variable type and branch to the appropriate processing routine:

```
TSTTYP  RST      32      ;Test TYPFLG
        JP       Z,STRING ;String Variable
        JP       M,INTGR  ;Integer Variable
        JP       PO,SNG   ;Single Precision Var.
        JP       NC,DBL   ;Double Precision Var.
```

(Note: to use RST 32, RAM addresses 4009H - 400BH must not have been changed from their original initialization. If there is a possibility that they will be, replace "RST 32" with "CALL 25D9H" in the above example.)

ERROR RECOVERY ROUTINE

One problem faced by those who have tried to interface with the ROM routines in the past is the fact that if an error occurs, any error, control is given to the BASIC interpreter, thousands of bytes away from the machine language program that was executing. For this reason, an error recovery routine had to be developed so that control would pass smoothly back to the calling routine for processing of the error.

Several conditions can cause an error in Level II when working with the math routines. One example is if the result of the addition, subtraction, multiplication, or division overflows the limits of the variable type. Errors can also occur when the RND function is called with a negative number as a parameter or with a number that exceeds the positive upper limit of an integer (32,767). Other problem areas include (but are by no means limited to): division by zero, undefined trigonometric call, type mis-matches, and other illegal function calls.

The following initialization routine should be placed at the beginning of the program, before any of the ROM routines are used:

```
TRAP      LD      HL,STACK      ;Point to program's STACK
          LD      (40E8H),HL    ;Replace Stack pointer in
                                ; error trap procedure.
          LD      HL,41A6H      ;Init trap vector
          LD      (HL),0C3H     ; to JP to Recovery rout.
          INC     HL
          LD      DE,RCVRY      ;Point to recovery rout.
          LD      (HL),E
          INC     HL
          LD      (HL),D
          LD      A,0C9H        ;Set-up RET in various
          LD      (41BEH),A     ; RAM vectors
          LD      (41C1H),A
          LD      (41D0H),A
          LD      (40F2H),A     ;This address <> zero!
          XOR     A             ;Clear A register
          LD      (409CH),A     ;This address = zero
```

The trap initialization routine is now complete. The following routine is executed when an error occurs in the ROM. It may be placed anywhere in the assembly language code.

```

RCVRY    LD      SP,STACK    ;Restore prog's stack
         LD      A,E         ;Pick-up Error number ptr
         SRA                     ;Divide error number by 2
         INC     A           ;Accumulator now contains
         .                  ; an error number which
         .                  ; matches on in Table B-1
         .                  ; of the Level II Manual
;Do error testing here...

```

DATA CONVERSION

In order to convert data values, it is necessary to initialize the addresses from 4080H - 408DH so that they contain the following values:

```

X'4080' = D6 00 6F 7C
X'4084' = DE 00 67 78
X'4088' = DE 00 47 3E
X'408C' = 00 C9

```

(This memory area is configured by the Level II initialization code and is used by the single precision division routine.)

The three-byte region at 4090H - 4092H must also be initialized if random number generation will be requested. The field contains the multiplicative mantissa constant used in the random number generator. The addresses are initialized as follows:

```

X'4090' = 40
X'4091' = E6
X'4092' = 4D

```

These initialization areas have been listed together since they can BOTH be set-up correctly through the execution of the following code:

```

INITRM  LD      DE,4080H      ;Point to RAM area
        LD      HL,18F7H      ;Point to data in ROM
        LD      BC,39         ;Bytes to move
        LDIR                      ;Initialize RAM

```

Data values in FPA1 can be converted from one TYPE to another, provided the resulting value does not overflow the requested type. The TYPFLG must have been previously set to the current type or the results will be unpredictable. This flag will be automatically revised upon completion of the conversion to the new type.

```

CINT    CALL    0A7FH         ;Convert FPA1 to Integer
CSNG    CALL    0AB1H         ;Convert FPA1 to Single
CDBL    CALL    0ADBH         ;Convert FPA1 to Double

```

These routines are general purpose, converting any type to the designated one. However, if the type is known, the routines below may be called to change the type. They provide slightly faster execution by skipping unneeded processing.

```

SNGINT  CALL    0ACCH         ;FPA1 from Integer to SNG
DBLSNG  CALL    0AB9H         ;FPA1 from DBL to SNG
SNGDBL  CALL    0AE3H         ;FPA1 from SNG to DBL

```

Also, if a value is stored at a different location in memory, it also can be converted by first setting TYPFLG and then calling the following routine:

```

        LD      HL,VALUE      ;Point HL to start of val.
HLSNG   CALL    0ACFH         ;Convert (HL) to SNG

```

**** WARNING **** If the above routines are called with the type flag (TYPFLG) set for an string value (3), an error will result. If the error recovery routine has not been implemented, control will be given to the BASIC interpreter.

MOVING DATA

In order to effectively use the math routines, it will be necessary to shift data in and out of the accumulators, registers, and user buffer regions. The routines described in the following paragraphs should be used for this purpose.

The routine at 09A4H will move the single precision value from FPA1 to the stack. The routine pops the return address off the stack, pushes the value onto the stack, replaces the return address, then returns to the calling routine.

```
STKFP1    CALL    09A4H        ;Put SNG in FPA1 onto stack
```

The stacked value can be recovered into the RFPA through the following sequence:

```
POPRFP    POP      BC          ;Recover 4123H-4124H
           POP      DE          ;Recover 4121H-4122H
```

If the value in the RFPA is to be placed back into FPA1, call this routine:

```
SNGFPA    CALL    09B4H        ;RFPA into FPA1
```

The next routine copies four (4) bytes from the region pointed to by the register pair HL and moves it to FPA1. The first byte is placed at 4121H, the second at 4122H, the third at 4123H, and the fourth at 4124H (LSB, LSB, MSB, EXP).

```
HLFPA1    CALL    09B1H        ;(HL) placed into FPA1
```

To manipulate single precision values, the ROM at times will copy FPA1 into the RFPA. This routine performs this function:

```
LDFPA1    CALL    09BFH        ;Load RFPA from FPA1
```

To eliminate the need of first moving a value from a buffer to FPA1 and then loading it into the RFPA, a single routine may be called. The first byte is placed in E, the second in D, the third in C, and the fourth in B:

```

                LD      HL,BUFF      ;Load beginning of buffer
                                ; containing value.
LDFPHL  CALL      09C2H      ;(HL) to RFPA

```

Once a routine has been called, one may want to transfer the resulting value in FPA1 to a buffer so that another value can be manipulated. This is accomplished by this routine:

```

                LD      HL,BUFF      ;Load beginning of buffer
FPAMEM  CALL      09CBH      ;FPA1 moved to (HL)

```

The next two routines are used to transfer double and single precision values from one location to another. The number of bytes to be moved is based on the value of the TYPFLG (four or eight).

This routine transfers bytes from the buffer pointed to by HL to the one pointed to by DE.

```

MOVTDE  CALL      09D2H      ;Move (HL) to (DE)

```

The perform the exact opposite of the above call, access the routine to move from buffer DE to buffer HL.

```

MOVTHL  CALL      09D3H      ;Move (DE) to (HL)

```

The last routine transfers either four or eight bytes from FPA1 to FPA2. However, the bytes transfered will begin at address 4127H which will not place them in the correct position for single precision values, but the transfer is correct for double precision values. Again, the number of bytes to be transfered is based on TYPFLG.

```

FP1FP2  CALL      09FCH      ;Move (FPA1) to (FPA2)

```

NUMERIC I/O IN ASCII

Before discussing the interfaces to the actual math routines in the next chapter, the methods for inputting and outputting numerical data in ASCII must be listed.

ASCII TO BINARY

The first task one must face is converting an ASCII buffer to its binary representation. The ROM provides a single routine to handle all of the I/O of this type. Fortunately, it is a very flexible routine, allowing the conversion of a wide variety of strings. This routine should be used to read in ASCII numbers and convert them to their binary equivalent for processing by the various math routines. To access this routine, perform the following:

```
LD      HL,BUFFER ;I/O Buffer
ASCBIN  CALL  0E65H ;Convert ASCII to binary
                        ; Default type = DBL
```

The buffer should begin with the first character to be converted. All numeric specifications legal in BASIC are recognized here. Therefore, the buffer can begin with a plus (+) or minus (-) sign. The numeric part of the number can be integer or real (contain a decimal point). It can also have scientific notation. The value can be followed by a TYPE specification (% = integer, ! = single precision, # = double precision) which will force the converted value to take on the specified type.

The numeric value can also be followed by "E" for single precision scientific notation or "D" for double precision scientific notation, but DO NOT mix explicit type and exponent type.

If the data value overflows the type specified, it will automatically be converted up to the minimum type necessary to contain the data value. If no type is specified, it will default to double precision.

The last character in the buffer MUST be either a colon (:) or a hex zero (00) for proper conversion.

The binary result produced is placed in the floating point accumulator (FPA1) as follows:

a. Integer result stored in addresses 4121H (LSB) and 4122H (MSB). The TYPFLG at 40AFH will be set to X'02' denoting an integer.

b. Single precision result is stored in addresses 4121H containing the least significant byte through 4124H containing the exponent. TYPFLG will be set to X'04' denoting a single precision value.

c. Double precision result is placed in FPA1 in addresses 411DH containing the LSB through 4124H containing the exponent. TYPFLG will be set to X'08' denoting a double precision value.

Another manner to access the same conversion routine is provided which defaults to a integer value unless an overflow occurs.

```
LD      HL,BUFFER ;Point to I/O buffer
                        ; value to convert
ASCINT  CALL  0E6CH ;Convert, Default type=INT
```

BINARY TO ASCII

Outputting of binary data can take on two forms, formatted and unformatted. There exists three calling routines, one for each type of variable.

For each of the following routines, the data is placed unformatted in an I/O buffer beginning at 4130H. The last character in the buffer is followed by a hex zero (00) to mark the end.

For integers, use this code:

```
LD      HL,VALUE ;VALUE = Value to convert
SAVINT  CALL  0A9AH ;Save integer to FPA1
BINASC  CALL  0FBDH ;Convert to ASCII
```

For single precision, the following code is used:

```
                LD      HL,VPTR    ;Load address where value is
                                ; stored.
SETSNG          CALL    0AEFH      ;Set TYPFLG to SNG
HLFPAE          CALL    09F7H      ;Value to FPA1
BINASC          CALL    0FBDH      ;Convert to ASCII
```

Double precision values are converted using this routine:

```
                LD      HL,VPTR    ;Load address of value
SETDBL          CALL    0AECH      ;Set TYPFLG to DBL
HLFPAE          CALL    09F7H      ;Value to FPA1
BINASC          CALL    0FBDH      ;Convert to ASCII
```

For formatted output, one can use the same code as above, but replace "CALL 0FBDH" in each one with the following:

```
FORMAT          LD      A,N        ;Load control codes
ASCUSG          CALL    0FBEH      ;Convert to formatted ASCII
                                ; in 4030H buffer.
```

The value of "N" depends on the format desired:

a. Set bit 3 (X'08') to place a plus (+) sign as the first character in the buffer if a positive value.

b. Set bit 4 (X'10') to place a dollar sign (\$) as the first character in the buffer.

c. Set bit 5 (X'20') to place an asterisk (*) as the first character in the buffer.

d. Set bit 6 (X'40') to place commas (,) at every third place of any assumed or explicit decimal point.

Any combination of the above is acceptable.

CHAPTER 3: ASSEMBLY INTERFACING

REGISTER UTILIZATION

Although some exceptions do exist, one should assume that ALL non-index registers will be used by the math routine calls. This includes registers AF, BC, DE, and HL. However, due to the fact that the BASIC part of the ROM was originally written in 8080 assembly language, the alternate registers are not used. Index register IX is only used by the I/O routines. Index register IY is not used at ANY time during execution of ROM code.

INTEGER MATH ROUTINES

As noted in the chapter on data formats, integers are represented in the TRS-80 as signed, sixteen bit values. These values are passed to the integer math routines in register pairs HL and DE. After performing the requested operation, the result is placed in the Floating Point Accumulator (FPA1) at addresses 4121H and 4122H (LSB/MSB). In addition, certain routines will place the result also in one of the register pairs. If this is the case, it will be specifically noted.

Addition of two signed integers is performed by loading one value into HL and the second into DE. The call is made to 0BD2H. The result is placed in FPA1, and if an overflow does not occur, it will also be placed in HL. After the return from the addition, the TYPFLG should be checked to determine whether an overflow has occurred. The ADDINT routine updates the TYPFLG to two (2) if the result is an integer and four (4) if the result overflowed and had to be expressed as a single precision value.

	LD	HL,VAL1	;First value to HL
	LD	DE,VAL2	;Second Value to DE
ADDINT	CALL	0BD2H	;(DE + HL) to FPA1
	RST	32	;Check type
	JP	PO,OVRFL	;If single prec., overflow

Subtraction of integers is performed in a similar manner. The values are placed in DE and HL, the call is made (this time to 0BC7H), the result of DE minus HL is returned in FPA1 and HL if no overflow, and the TYPFLG is adjusted automatically depending on the result.

```

                LD      HL,VAL1    ;First Value to HL
                LD      DE,VAL2    ;Second Value to DE
SUBINT          CALL    0BC7H      ;(DE - HL) to FPA1
                RST     32         ;Test overflow
                JP      PO,OVRFL   ;Jump if overflow

```

Multiplication is performed in the above manner also.

```

                LD      HL,VAL1    ;First Val. to Reg. HL
                LD      DE,VAL2    ;Second Val. to Reg. DE
MULINT          CALL    0BF2H      ;(DE * HL) to FPA1
                RST     32         ;Overflow?
                JP      PO,OVRFL   ;If SNG, overflow

```

Division of two integers is handled in a totally different fashion. Both values are first converted to single precision prior to performing the division. The result is always a single precision value in FPA1 with the TYPFLG set to four (4). If modular division is desired (true integer division), perform a call to the INT function following the call to 2490H.

```

                LD      HL,VAL1    ;Divisor to HL
                LD      DE,VAL2    ;Dividend to DE
DIVINT          CALL    2490H      ;(DE / HL) to FPA1
INT             CALL    0B37H      ;Take integer of (DE / HL)
                                ; (Optional)

```

The last routine for integers is the comparison routine. Registers DE and HL are set-up in the same way. The call is then made to 0A39H. This routine returns a value in register A or -1, 0, or +1 depending on whether the contents of HL are less than, equal to, or greater than the contents of DE.

```

                LD      HL,VAL1    ;1st Value to Reg. HL
                LD      DE,VAL2    ;2nd Value to Reg. DE
CPRINT         CALL    0A39H      ;CP HL to DE
                JP      M,LESS     ;HL < DE
                JP      Z,EQUAL    ;HL = DE
                JP      P,PLUS     ;HL > DE

```

SINGLE PRECISION MATH ROUTINES

Single precision numbers require four bytes for storage as noted earlier in the Data Formats chapter. These values are passed to the single precision routines in the RFPA (registers BCDE) and FPAL. Several methods can be used to move the data to the RFPA and FPAL, many of which were presented in the chapter on data manipulation. In the following examples, only a few will be noted.

The result of the addition of two single precision values is placed in FPAL.

```

SETSNG         CALL    0AEFH      ;Set TYPFLG to SNG (4)
                LD      HL,BUF1    ;Point to 1st value
HLFPAL         CALL    09B1H      ;Move buffer to FPAL
                LD      HL,BUF2    ;Point to 2nd value
LDDPHL         CALL    09C2H      ;Load RFPA from buffer
ADDSNG         CALL    0716H      ;(RFPA + FPAL) to FPAL

```

The ROM has two dedicated routines for addition of single precision values. The first adds 0.5 to the contents of the FPAL. The other adds the four-byte value in the buffer pointed to by HL to the value in FPAL.

```

ADHALF         CALL    0708H      ;0.5 + FPAL to FPAL

                LD      HL,BUFF1    ;Point to value
ADDHL          CALL    070BH      ;(HL) + FPAL to FPAL

```

Subtraction of single precision values is handled by two routines. The first, like addition, utilizes the RFPA and FPAL. This call subtracts FPAL from RFPA, placing the result in FPAL.

```

SETSNG    CALL    0AEFH      ;Set TYPFLG to SNG (4)
           LD      HL,BUF1    ;Point to 1st value
HLFPA1    CALL    09B1H      ;Move buffer to FPA1
           LD      HL,BUF2    ;Point to 2nd value
LDDPHL    CALL    09C2H      ;Load RFPA from buffer
SUBSNG    CALL    0713H      ;(RFPA - FPA1) to FPA1

```

An alternate method is to subtract the value pointed to by HL from FPA1. Linkage as follows:

```

           LD      HL,BUF1    ;Point to SNG Val. w/ HL
SUBHL     CALL    0710H      ;(HL) - FPA1 to FPA1

```

Single precision multiplication is performed using the same registers and accumulator as addition. A call is then made to 0847H.

```

SETSNG    CALL    0AEFH      ;Set TYPFLG to SNG (4)
           LD      HL,BUF1    ;Point to 1st value
HLFPA1    CALL    09B1H      ;Move buffer to FPA1
           LD      HL,BUF2    ;Point to 2nd value
LDDPHL    CALL    09C2H      ;Load RFPA from buffer
MULSNG    CALL    0847H      ;(RFPA * FPA1) to FPA1

```

Another routine multiplies the single precision value in FPA1 by ten (10), placing the result in FPA1.

```

MUL10     CALL    093EH      ;10.0 * FPA1 to FPA1

```

Single precision division divides the value in RFPA by the value in FPA1. Remember that the RAM area (4080H - 408DH) described in Chapter 2 MUST BE INITIALIZED BEFORE THE DIVISION ROUTINES ARE USED!

```

SETSNG    CALL    0AEFH      ;Set TYPFLG to SNG (4)
           LD      HL,BUF1    ;Point to 1st value
HLFPA1    CALL    09B1H      ;Move buffer to FPA1
CKRZMP    CALL    0955H      ;Check FPA1 for M,Z,P
           JP      Z,DIVZER    ;Divide by zero error
           LD      HL,BUF2    ;Point to 2nd value

```

```

LDDPHL    CALL    09C2H    ;Load RFPA from buffer
DIVSNG    CALL    08A2H    ;(RFPA / FPA1) to FPA1

```

The above routine tests the divisor for zero. If the test is not made, the divisor IS zero (remember Murphy's Law?!?), and the error trap has not been implemented, ...

```

?/0 ERROR
READY
>

```

An alternate linkage, assuming that the dividend is in FPA1, would be as follows:

```

SETSNG    CALL    0AEFH    ;Set TYPFLG to SNG (4)
STKFPA1   CALL    09A4H    ;FPA1 (Dividend) to Stack
           LD      HL,BUF1  ;Point to 1st value
HLFPA1    CALL    09B1H    ;Move buffer to FPA1
CKRMZP    CALL    0955H    ;Check for -, 0, +
           JP      Z,DIVERR ;Divide by zero error
POPFPA    CALL    08A0H    ;(STACK / FPA1) to FPA1

```

A divide by ten can be performed by loading FPA1 with the dividend and then issuing the following call:

```

DIV10     CALL    0897H    ;FPA1 / 10.0 to FPA1

```

To compare two single precision values, one in RFPA and the other in FPA1, use the following code:

```

SETSNG    CALL    0AEFH    ;Set TYPFLG to SNG (4)
           LD      HL,BUF1  ;Point to 1st value
HLFPA1    CALL    09B1H    ;Move buffer to FPA1
           LD      HL,BUF2  ;Point to 2nd value
LDDPHL    CALL    09C2H    ;Load RFPA from buffer
CPRSNG    CALL    0A0CH    ;CP FPA1 to RFPA
           JP      M,LESS   ;FPA1 < RFPA
           JP      Z,EQUAL  ;FPA1 = RFPA
           JP      P,MORE   ;FPA1 > RFPA

```

Also note that register A will contain -1, 0, or +1 depending on the outcome of the compare.

DOUBLE PRECISION MATH ROUTINES

The five operations--addition, subtraction, multiplication, division, and comparison--are performed using FPA1 (411DH - 4124H) and FPA2 (4127H - 412EH). If the operation requested results in a value that overflows the upper limit of a double precision variable (+1.701411834544556D+38), an overflow error will result. The calling program can recover by using the error recovery procedure outlined in chapter 2.

To add two double precision values, use:

```
SETDBL  CALL  0AECH      ;Set TYPFLG for DBL (8)
        LD    HL,BUF1    ;Point to first value
HLFPAE  CALL  09F7H      ;Move to FPA1
        LD    HL,BUF2    ;Point to 2nd value
        LD    DE,4127H   ;Point to FPA2
MOVTDE  CALL  09D2H      ;Move to FPA2
ADDDBL  CALL  0C77H      ;(FPA1 + FPA2) to FPA1
```

To subtract two double precision numbers, use:

```
SETDBL  CALL  0AECH      ;Set TYPFLG for DBL (8)
        LD    HL,BUF1    ;Point to first value
HLFPAE  CALL  09F7H      ;Move to FPA1
        LD    HL,BUF2    ;Point to 2nd value
        LD    DE,4127H   ;Point to FPA2
MOVTDE  CALL  09D2H      ;Move to FPA2
SUBDBL  CALL  0C70H      ;(FPA1 - FPA2) to FPA1
```

Multiplication of double precision numbers can be performed using:

```
SETDBL  CALL  0AECH      ;Set TYPFLG for DBL (8)
        LD    HL,BUF1    ;Point to first value
HLFPAE  CALL  09F7H      ;Move to FPA1
        LD    HL,BUF2    ;Point to 2nd value
        LD    DE,4127H   ;Point to FPA2
MOVTDE  CALL  09D2H      ;Move to FPA2
MULDBL  CALL  0DA1H      ;(FPA1 * FPA2) to FPA1
```

To divide FPA1 by FPA2, this linkage is employed (which includes error checking for a divisor of zero):

```

SETDBL  CALL  0AECH      ;Set TYPFLG for DBL (8)
        LD    HL,BUF1    ;Point to first value
HLFPAE  CALL  09F7H      ;Move to FPA1
        LD    HL,BUF2    ;Point to 2nd value
        LD    DE,4127H   ;Point to FPA2
MOVTDE  CALL  09D2H      ;Move to FPA2
        LD    A,(412EH)  ;Check FPA2 for zero
        OR    A          ;Set Z flag on zero
        JP    Z,DIVZER   ;Divide by zero error
DIVDBL  CALL  0DE5H      ;(FPA1 / FPA2) to FPA1

```

The comparison of two double precision values is performed as follows:

```

SETDBL  CALL  0AECH      ;Set TYPFLG for DBL (8)
        LD    HL,BUF1    ;Point to first value
HLFPAE  CALL  09F7H      ;Move to FPA1
        LD    HL,BUF2    ;Point to 2nd value
        LD    DE,4127H   ;Point to FPA2
MOVTDE  CALL  09D2H      ;Move to FPA2
CPRDBL  CALL  0A78H      ;Compare FPA1 to FPA2
        JP    M,LESS     ;FPA1 < FPA2
        JP    Z,EQUAL    ;FPA1 = FPA2
        JP    P,MORE     ;FPA1 > FPA2

```

INTERFACING TO FUNCTIONS

This section will discuss the methods of interfacing to the math functions (ABS, ATN, COS, EXP, FIX, INT, LOG, POWER, RND, SGN, SIN, SQR, TAN). It is important to note that all of these functions, except ABS, FIX, INT, and RND, operate exclusively on single precision data. Therefore, in most cases, the floating point accumulator (FPA1) will be used to provide the argument to the function and will be the destination of the result. The POWER function is also an exception since it requires two arguments and also uses the RFPA as well as FPA1.

ABS(X) - Absolute Value Function

The ABS(X) function will return the absolute value of an integer, single precision, or double precision argument loaded into FPA1. The following linkages are used for ABS:

ABSINT	LD	HL,VALUE	;Put integer value into HL
SAVINT	CALL	0A9AH	;Place into FPA1
ABS	CALL	0977H	;Take Absolute value.
ABSSNG	LD	HL,VALUE	;Point to SNG value
HLFPA1	CALL	09B1H	;Value to FPA1
SETSNG	CALL	0AEFH	;TYPFLG set for SNG (4)
ABS	CALL	0977H	;Take Absolute Value
ABSDBL	CALL	0AECH	;Set TYPFLG for DBL (8)
	LD	HL,VALUE	;Point to DBL Value
HLFPAE	CALL	09F7H	;Value to FPA1
ABS	CALL	0977H	;Take absolute Value

ATN(X) - Arctangent Function

Arctangent is a trigonometric function which requires a single precision argument loaded into FPA1. This argument is a number which represents the value of the TAN function at the specified angle, the exact value of which will be returned by the ATN function. (i.e., $\text{TAN}(X)=Y$, $\text{ATN}(Y)=X$).

	LD	HL,BUF1	;Point to data value
HLFPA1	CALL	09B1H	;Move to FPA1
SETSNG	CALL	0AEFH	;TYPFLG to SNG (4)
ATN	CALL	15BDH	;ATN(X) returned in FPA1

COS(X) - Cosine Function

COS is another trigonometric function that requires the argument to be a single precision number loaded into FPA1. The result is returned in FPA1. The cosine is computed by using the trig identity $\text{COS}(X) = \text{SIN}(X+\text{PI}/2)$. The value "X+PI/2" is first computed, then its SIN is taken. The constant PI/2 is evaluated as 1.5708. Linkage to this function is as follows:

	LD	HL,BUF1	;Point to data value
HLFPAL	CALL	09B1H	;Move to FPAL
SETSNB	CALL	0AEFH	;TYPFLG to SNG (4)
COS	CALL	1541H	;COS(X) returned in FPAL

EXP(X) - Exponential Function

The exponential function raises the value of "e" to the power, X. The argument, X, is a single precision value placed into FPAL. It is important to note that an overflow error can occur if the value of X is too large. Any value exceeding 87.3363 (126 divided by 1.4427 which is the Log base 2 of e) is too large and will result in overflow. Any value less than -88.7225 (-128 divided by 1.4427) will return a result of zero.

	LD	HL,BUF1	;Point to data value
HLFPAL	CALL	09B1H	;Move to FPAL
SETSNB	CALL	0AEFH	;TYPFLG to SNG (4)
EXP	CALL	1439H	;EXP(X) returned in FPAL

FIX(X) - Truncation Function

The FIX function will truncate a data value (i.e., if $Y = \text{FIX}(+2.76)$, $Y = +2$; if $Y = \text{FIX}(-2.76)$, $Y = -2$). This function accepts arguments that are integer, single precision, or double precision; the TYPFLG indicates the type of argument. However, FIXing an integer performs no useful function.

FIXINT	LD	HL,VALUE	;Put integer value into HL
SAVINT	CALL	0A9AH	;Place into FPAL
FIX	CALL	0B26H	;FIX value.
FIXSNG	LD	HL,VALUE	;Point to SNG value
HLFPAL	CALL	09B1H	;Value to FPAL
SETSNB	CALL	0AEFH	;TYPFLG set for SNG (4)
FIX	CALL	0B26H	;FIX Value
FIXDBL	CALL	0AECH	;Set TYPFLG for DBL (8)
	LD	HL,VALUE	;Point to DBL Value
HLFPAL	CALL	09F7H	;Value to FPAL
FIX	CALL	0B26H	;FIX Value

INT(X) - Greatest Integer Function

The INT function finds the greatest integer not exceeding the argument (i.e., if $Y = \text{INT}(+2.76)$, $Y = +2$; if $Y = \text{INT}(-2.76)$, $Y = -3$). All types of numeric types are accepted; TYPFLG should be set to the type to be INTed.

INTINT	LD	HL,VALUE	;Put integer value into HL
SAVINT	CALL	0A9AH	;Place into FPA1
INT	CALL	0B37H	;INT value.
INTSNG	LD	HL,VALUE	;Point to SNG value
HLFPA1	CALL	09B1H	;Value to FPA1
SETSNG	CALL	0AEFH	;TYPFLG set for SNG (4)
INT	CALL	0B37H	;INT Value
INTDBL	CALL	0AECH	;Set TYPFLG for DBL (8)
	LD	HL,VALUE	;Point to DBL Value
HLFPAE	CALL	09F7H	;Value to FPA1
INT	CALL	0B37H	;INT Value

LOG(X) - Natural Logarithm Function

The LOG function returns the Log base e of the argument passed in FPA1. LOG cannot be used on negative values. The following interface, with error-checking, performs the LOG function:

	LD	HL,VAL1	;Point to data value
HLFPA1	CALL	09B1H	;Move value to FPA1
SETSNG	CALL	0AEFH	;Set TYPFLG for SNG (4)
CKRMZP	CALL	0955H	;Check for -, 0, +
	CALL	M,ERROR	;Branch if Negative
LOG	CALL	0809H	;Take LOG
	.		
	.		
	.		
ERROR	CALL	0977H	;Take ABS(FPA1)
	LD	HL,ERRMSG	;Point to error msg
MSGOUT	CALL	2B75H	;Output error msg to CRT
	RET		;Return to take LOG(ABS(X))
ERRMSG	DEFM	'LOG Error - Negative Argument'	
	NOP		;End of Msg delimiter

The algorithm used to calculate the LOG is as follows:

```
LET      M = Mantissa(X)
          C = SQRT(2)/2
          R = Exponent(X)
          C1 = .598979
          C2 = .961471
          C3 = 2.88539
          C4 = .693147

          M = (M-C)/(M+C)
          K = M * M
          LOG(X) = (((C1*K+C2)*K+C3)*M+R-0.5)*C4
```

POWER - Raise X to the power of Y

This routine is accessed from BASIC using the up-arrow key. The value of X must be a single precision value loaded onto the STACK. The value of Y can be of type single, double, or integer and is placed in FPA1. The TYPFLG is set according to the value of Y. Linkage reflecting a single precision Y is as follows:

```
LD      HL,RETADR ;Set return address
PUSH    HL        ;Ret addr. to top of stack
LD      HL,BUFX   ;Point to X
HLFPA1  CALL 09B1H ;Move to FPA1
STKFP1  CALL 09A4H ;Place on Stack
SETSNG  CALL 0AEFH ;Set TYPFLG to SNG (4)
LD      HL,BUFX   ;Point to X
TOFPA1  CALL 09B1H ;Move to FPA1
JP      13F2H     ;Take X power Y
```

As can be noted from the above linkage, entry is made to the POWER routine via a JUMP instruction with a return address pushed onto the stack as the first operation. This must occur in this fashion since the data value X must be at the top of the stack upon entry to the POWER routine.

X POWER Y is computed by setting $EXP(Z) = X \text{ POWER } Y$ then solving for Z from $Z = Y * LOG(X)$. The EXP function then uses Z as its argument to calculate $EXP(Z)$, the answer.

RND(X) - Random Number Generation Function

The RND function provides for the generation of a random number as its result. Depending on the value of X, it will return either a single precision value in the range zero through one (0.0 to 1.0 when X=0), or an integer value between one (1) and the value of X (which must be between one and 32,767 or an error will result).

The RND routine uses RAM memory at addresses 4090H to 4092H which must be initialized to X'40 E6 4D', which can be accomplished by using the initialization routine described in Chapter 2. Addresses 40AAH to 40ACH are used as part of the "seed" value. After initialization, do not disturb these areas!

The random number generated can be "randomized" through a call to the RANDOM command:

```
RANDOM    CALL    01D3H        ;Randomize seed
```

The RANDOM routine sets a single byte in the "seed" field from the Z80 refresh register (LD A,R). This call only modifies the contents of reg. A.

The following code will return, in FPA1, a single precision random number between zero and one:

```
SETSNG    CALL    0AEFH        ;Set TYPFLG to SNG (4)
ZERFPA    CALL    0778H        ;Zero FPA1
RND       CALL    14C9H        ;Generate random number
```

This linkage will return an integer random number between one and the integer value passed to the RND routine in FPA1:

	LD	HL,MAXVAL	;Generate number between one ; and MAXVAL
SAVINT	CALL	0A9AH	;Save in FPAL. TYPFLG=INT
RND	CALL	14C9H	;Generate random number
CINT	CALL	0A7FH	;Convert to integer

SGN(X) - Determine Sign of Argument

The SGN function determines the algebraic sign of integer, single or double precision arguments. The argument is loaded into FPAL. Depending on whether the argument is negative, zero or positive, the result is a value of -1, 0, or +1 expressed as an integer value in FPAL (4121H - 4122H). The TYPFLG is automatically set to 2 to indicate an integer value. Be aware that the value one loads into FPAL will be overwritten by the result. Linkage to the SGN routine for each argument type is as follows:

SGNINT	LD	HL,VALUE	;Put integer value into HL
SAVINT	CALL	0A9AH	;Place into FPAL
SGN	CALL	098AH	;Return Sign of Value
SGNSNG	LD	HL,VALUE	;Point to SNG value
HLFPAL	CALL	09B1H	;Value to FPAL
SETSNG	CALL	0AEFH	;TYPFLG set for SNG (4)
SGN	CALL	098AH	;Return Sign of Value
SGNDBL	CALL	0AECH	;Set TYPFLG for DBL (8)
	LD	HL,VALUE	;Point to DBL Value
HLFPAL	CALL	09F7H	;Value to FPAL
SGN	CALL	098AH	;Return Sign of Value

SIN(X) - Sine Function

SIN is a trigonometric function which requires a single precision argument placed in FPAl. The resultant sine is returned in FPAl.

	LD	HL,BUF1	;Point to data value
HLFPAl	CALL	09B1H	;Move to FPAl
SETSNB	CALL	0AEFH	;TYPFLG to SNG (4)
SIN	CALL	1547H	;SIN(X) returned in FPAl

SQR(X) - Square Root Function

The SQR function takes the square root of a positive single precision value located in FPAl. An error will occur if it is called with a negative value in the floating point accumulator. For this reason, the following linkage takes the absolute value of any negative numbers before making the call.

	LD	HL,BUF1	;Point to data value
HLFPAl	CALL	09B1H	;Move to FPAl
SETSNB	CALL	0AEFH	;TYPFLG to SNG (4)
CKRMZP	CALL	0955H	;Check FPAl for -, 0, +
ABS	CALL	M,0977H	;Take ABS(-X)
SQR	CALL	13E7H	;SQR(X) returned in FPAl

TAN(X) - Tangent Function

The TAN function also requires a single precision value in FPAl as its argument. The resultant is placed in FPAl. The tangent is computed using the trig identity "TAN(X) = SIN(X) / COS(X)." The linkage for TAN is as follows:

	LD	HL,BUF1	;Point to data value
HLFPAl	CALL	09B1H	;Move to FPAl
SETSNB	CALL	0AEFH	;TYPFLG to SNG (4)
TAN	CALL	15ABH	;TAN(X) returned in FPAl



CHAPTER 4: DISASSEMBLY OF ROM MATH ROUTINES

In this chapter, one will find a commented disassembly of the Level II BASIC ROM mathematics routines. However, a few important points must be made.

First of all, the ROM code is the property of Microsoft and is protected by their copyright. For this reason, it is impossible to provide a complete disassembly of their code without violating their rights. For this reason, the publisher decided to provide the hex address of the instruction, the operator, and the comments. The hex object code and the operands are omitted.

If the reader is an owner of a TRS-80, he is then able to procure the full disassembly by using a machine language disassembler or the BASIC language disassembler provided in Appendix C. Space has been provided so that the operands can be written in next to the operators to provide a commented listing that can be used for reference.

Secondly, since a full interfacing guide is provided in the first three chapters, it is unnecessary to refer to this listing in order to interface with the routines. Nevertheless, this chapter has been provided for those that may be curious as to the manner in which the ROM operates.

Math Routines: Disassembly

```

;*****
;SINGLE PRECISION ADDITION & SUBTRACTION
;      0708 -> 0.5  + FPA1 -> FPA1
;      070B -> (HL) + FPA1 -> FPA1
;      0710 -> (HL) - FPA1 -> FPA1
;      0713 -> RFPA - FPA1 -> FPA1
;      0716 -> RFPA + FPA1 -> FPA1
;*****
0708 LDHALF LD          ;LOAD RFPA WITH 0.5
070B Z070B CALL        ;LOAD REAL VALUE AT (HL) INTO FPA1
070E JR
0710 Z0710 CALL        ;LOAD RFPA WITH (HL)
0713 SUBSNG CALL      ;MAKE FPA1 NEGATIVE
0716 ADDSNG LD        ;TEST RFPA FOR ZERO (VAR2)
0717 OR          ;& RETURN IF SO
0718 RET
0719 LD          ;IF FPA1 (VAR1) IS ZERO,
071C OR          ;PUT VAR2 -> FPA1 & EXIT
071D JP
0720 SUB
0721 JR          ;JUMP IF VAR1 > VAR2
0723 CPL          ;CVRT EXP DIFF TO +
0724 INC

;*****
;EXCHANGE VAR1 IN FPA1 WITH VAR2 IN RFPA
;*****
0725 EX          ;PLACE VAR1 ON STACK
0726 CALL        ;PUT REAL ONTO STACK
0729 EX
072A CALL        ;RFPA (VAR2) -> FPA1
072D POP         ;RECOVER VAR1 IN RFPA
072E POP

;*****
;IF DIFF IN EXP > 10**7, DON'T BOTHER TO ADD
;*****
072F Z072F CP      ;2**25 APPX 10**7
0731 RET          ;RET WITH FPA1 = LARGER VAL
0732 PUSH
0733 CALL        ;TURN ON SIGN BITS
0736 LD
0737 POP
0738 CALL        ;DIVIDE VAR1 MANTISSA BY THE
                ;DIFFERENCE IN EXPONENTS
073B OR          ;TEST RESULT OF SIGNS
073C LD
073F JP          ;JUMP IF SIGN BITS WERE .NE.
0742 CALL        ;ADD MANTISSA (HL) TO EDC
0745 JP          ;INC EXPON IF ADD OVERFLOWED
0748 INC         ;PT TO EXPONENT & ADD 1 DUE
0749 INC         ;TO CARRY ON MANTISSA ADD

```

Math Routines: Disassembly

```

074A      JP          ;OVERFLOW ERROR
074D      LD          ;INIT ROTATE LOOP TO 1 TO
074F      CALL        ;DIV MANTISSA BY 2 FOR
0752      JR          ;EXPONENT INCREASE
;*****
; SUBTRACT ONE NBR FROM ANOTHER WHEN 'SIGNS' WERE DIFFERENT
;*****
0754      Z0754      XOR          ;COMPLEMENT EXPON IN RFPA
0755      SUB          ;AND SET THE CARRY FLAG
0756      LD
;*****
;SUBTRACT MANTISSA IN RFPA FROM MANTISSA IN FPA1
;*****
0757      LD          ;SUB LOW BYTE
0758      SBC
0759      LD
075A      INC          ;PT TO 4122
075B      LD          ;SUB MID BYTE
075C      SBC
075D      LD
075E      INC          ;PT TO 4123
075F      LD          ;SUB HIGH BYTE
0760      SBC
0761      LD
0762      Z0762      CALL        ;TWOS COMPLEMENT THE RFPA
;*****
;          SNGL PREC NORMALIZATION ROUTINE
;*****
0765      NRMLZS      LD          ;XFER EXPON & LOW BYTE SINCE
0766      LD          ;NEED THE REGS B & E
0767      XOR          ;SET SHIFT COUNTER TO ZERO
0768      SSHFT8      LD          ;SET SHIFT COUNTER FROM ACCUMULATOR
0769      LD          ;IF HIGH BYTE IS 0
076A      OR          ;SHIFT LEFT 8 BITS
076B      JR
076D      LD          ;LEFT SHIFT THE MANTISSA 8 BITS
076E      LD          ;BY JUGGLING THE REGISTERS
076F      LD
0770      LD
0771      LD          ;P/U THE COUNTER & COUNT DOWN 8
0772      SUB
0774      CP          ;IF MANTISSA WAS NOT SHIFTED
0776      JR          ;OUT OF RFPA, THEN CYCLE
;*****
; MANTISSA WAS ALL ZEROES, ZERO THE RESULT
;*****
0778      ZERFPA      XOR          ;ZERO FPA1
0779      Z0779      LD
077C      RET
;*****

```

Math Routines: Disassembly

```

; CONTINUE TO NORMALIZE WITH SINGLE BIT SHIFTS
;*****
077D SSHFT1 DEC ;REDUCE THE SHIFT COUNTER
077E ADD ;MUL 'E' & 'B' BY 2 (HL SAVED B&E)
077F LD ;SHIFT MID BYTE (MUL BY 2)
0780 RLA
0781 LD
0782 LD ;SHIFT HIGH BYTE (MUL BY 2)
0783 ADC
0784 LD
0785 SCHKP JP ;HAS 1 SHIFTED INTO SIGN?
;*****
; A 1 HAS SHIFTED INTO THE SIGN POSITION, CLEANUP THE NUM
;*****
0788 LD ;P/U THE SHIFT COUNTER
0789 LD ;SET THE LOW BYTE
078A LD ;P/U THE EXPON MODIFIED
078B Z078B OR ;SEE IF ANY SHIFTING HAD OCCURRED
078C JR ;I.E. WAS IT ALREADY NORMALIZED?
;*****
; NUMBER HAD TO BE SHIFTED TO BE NORMALIZED. CORRECT THE EXPON
;*****
078E LD ;ADD THE SHIFT COUNTER (WHICH
0791 ADD ;IS A NEGATIVE VALUE) TO
0792 LD ;FPA1'S EXPONENT
0793 JR ;ZERO FPA1 IF SHIFT < EXPON
0795 RET ;RET WITH ZERO IF SHIFT = EXPON
;*****
; SHIFT WAS GREATER THAN EXPONENT, CONTINUE
;*****
0796 Z0796 LD ;P/U THE RESULT EXPONENT
0797 Z0797 LD
079A OR ;IS BIT 7 SET?
079B CALL ;INC RFPA BY 1 IF IT IS!
079E LD ;P/U THE FPA1 EXPON -> B
079F INC ;THEN PT TO 4125 FOR 'SIGN'
07A0 LD ;P/U THE PROCESSED SIGN BITS
07A1 AND ;& STRIP OFF ALL BUT THE SIGN BIT
;*****
; REPLACE THE SIGN POSITION IN RFPA (WHICH IS ALWAYS A 1
;AFTER NORMALIZATION) WITH THE CORRECT SIGN
;*****
07A3 XOR ;PUT CORRECT SIGN INTO RFPA
07A4 LD
07A5 JP ;REPLFPA1 WITH ADD/SUB RESULT
;*****
; INCREASE THE RFPA BY 1
;*****
07A8 Z07A8 INC ;INC LOW BYTE
07A9 RET

```

Math Routines: Disassembly

```

07AA          INC          ;INC MID BYTE IF 'CARRY'
07AB          RET
07AC          INC          ;INC HIGH BYTE IF 'CARRY'
07AD          RET
07AE          LD           ;PUT THE 1 BACK INTO THE 'SIGN' BIT
07B0          INC          ;& INC THE EXPONENT
07B1          RET
07B2  OVERR  LD           ;OVERFLOW ERROR
07B4          JP
;*****
; ADD MANTISSA POINTED TO BY HL TO MANTISSA IN REGS EDC
;*****
07B7  Z07B7  LD           ;ADD LOW BYTE
07B8          ADD
07B9          LD
07BA          INC          ;PT TO MID BYTE AND ADD
07BB          LD
07BC          ADC
07BD          LD
07BE          INC          ;PT TO HIGH BYTE AND ADD
07BF          LD
07C0          ADC
07C1          LD
07C2          RET
;*****
; TWO'S COMPLEMENT OF RFPA
;*****
07C3  S2SCMP LD           ;INVERT THE 'SIGN' BIT RESULT
07C6          LD
07C7          CPL
07C8          LD
07C9          XOR          ;SET L TO ZERO
07CA          LD
07CB          SUB          ;COMP 'B' & SET C-FLAG
07CC          LD
07CD          LD           ;ZERO ACCUM
07CE          SBC          ;COMP LOW BYTE
07CF          LD
07D0          LD           ;ZERO ACCUM
07D1          SBC          ;COMP MID BYTE
07D2          LD
07D3          LD           ;ZERO ACCUM
07D4          SBC          ;COMP HIGH BYTE
07D5          LD
07D6          RET
;*****
; PERFORM A RIGHT CIRCULAR SHIFT OF A MANTISSA
; BASED ON THE VALUE IN REG A
;*****
07D7  SSHTR  LD           ;INIT B TO ZERO

```

Math Routines: Disassembly

```

07D9  SSHTR8  SUB          ;TEST FOR 8 OR MORE BITS
07DB          JR          ;JUMP IF SHIFT < 8
07DD          LD          ;JUGGLE THE REGS
07DE          LD
07DF          LD
07E0          LD
07E2          JR
07E4  Z07E4  ADD          ;ADD BACK THE 8 + 1 MORE
07E6          LD          ;INIT THE SHIFT COUNTER
07E7  SSHTR1 XOR          ;ZERO A & REDUCE COUNTER
07E8          DEC
07E9          RET          ;FINISHED WHEN CTR RUNS OUT
07EA          LD          ;SHIFT HIGH BYTE
07EB  Z07EB  RRA
07EC          LD
07ED          LD          ;SHIFT MID BYTE
07EE          RRA
07EF          LD
07F0          LD          ;SHIFT LOW BYTE
07F1          RRA
07F2          LD
07F3          LD          ;SHIFT EXPONENT
07F4          RRA
07F5          LD
07F6          JR

;*****
;LOG ROUTINE DATA VALUES
;*****
07F8  Z07F8  DEFW        0          ;REAL 1.0
07FA          DEFW        810
07FC  Z07FC  DEFB         3          ;3 LOG CONSTANTS FOLLOW
07FD          DEFW        56AAH      ;0.598979
07FF          DEFW        8019H
0801          DEFW        22F1H      ;0.961471
0803          DEFW        8076H
0805          DEFW        0AA56H     ;2.88539
0807          DEFW        8238H

;*****
;PROCESS LOG(X)
;      ALGORITHM AS FOLLOWS:
;LET M = MANTISSA(X) : C=SQRT(2)/2 : R = EXPONENT(X)
;      M = (M-C)/(M+C) : K = M * M
;      LOG(X) = (((C1*K+C2)*K+C3)*M+R-0.5)*C4
;      C1=.598979 C2=.961471 C3=2.88539 C4=.693147
;*****
0809  LOG     CALL          ;CHECK MINUS,ZERO,PLUS
080C          OR          ;ILLEGAL FUNCTION CALL
080D          JP          ;IF NEG ARGUMENT
0810          LD          ;PLACE FPA1 EXPONENT -> A
0813          LD

```

Math Routines: Disassembly

```

0814      LD          ;0.707107 (SQRT(2)/2)
0817      LD
081A      SUB          ;SUB OFF X'80' FROM EXP
081B      PUSH          ;& SAVE RESULT
081C      LD          ;PLACE X'80' AS NEW EXPONENT
081D      PUSH          ;SAVE SQRT(2)/2
081E      PUSH
081F      CALL          ;ADD SINGLE PRECISION
0822      POP           ;RECOVER SQRT(2)/2
0823      POP
0824      INC           ;& MULTIPLY BY 2
0825      CALL          ;RFPA/FPA1
0828      LD           ;SUBTRACT FPA1 FROM 1.0
082B      CALL
082E      LD           ;POINT TO TABLE VALUES
0831      CALL          ;& PROCESS THE SERIES
0834      LD           ;-.0.5
0837      LD
083A      CALL          ;SUBTRACT 0.5
083D      POP           ;RECOVER EXPONENT EXCESS
083E      CALL
0841      Z0841      LD           ;0.693147 = LOG(2)
0844      LD
;*****
;MULTIPLICATION, SINGLE PRECISION
;*****
0847      MULSNG      CALL          ;IMMEDEIATE RET IF VARIABLE = 0
084A      RET
084B      LD
084D      CALL          ;ADD BOTH EXPONENTS
0850      LD           ;STORE BYTE WITH THE SIGN BIT
0851      LD
0854      EX           ;SAVE TWO LOW ORDER BYTES
0855      LD           ;OF THE MANTISSA
0858      LD           ;ZERO RFPA
085B      LD
085C      LD
085D      LD           ;INIT RETURN TO NORMALIZE
0860      PUSH
0861      LD           ;INIT RET TWICE FOR
0864      PUSH          ;2ND & 3RD BYTES
0865      PUSH
0866      LD           ;PT TO LOW ORDER BYTE
0869      Z0869      LD           ;P/U BYTE FROM FPA1
086A      INC          ;& PT TO NEXT BYTE
086B      OR
086C      JR           ;JUMP IF BYTE IS ALL ZERO
086E      PUSH          ;SAVE PTR TO FPA1 BYTE
086F      LD           ;INIT FOR 8 BITS
0871      Z0871      RRA          ;PASS BIT TO CARRY FLAG

```

Math Routines: Disassembly

```

0872      LD          ;SAVE TEST BYTE
0873      LD          ;PUT HIGH ORDER BYTE -> A
0874      JR          ;BYPASS ADD IF TEST BYTE BIT=0
0876      PUSH        ;SAVE TEST BYTE
0877      LD          ;P/U 2 LOW ORDER BYTES
087A      ADD         ;ADD MANTISSA (RFPA)
087B      EX
087C      POP         ;RESTORE TEST BYTE
087D      LD          ;ADD IN THE HIGH ORDER BYTE
0880      ADC
0881  Z0881  RRA       ;ROTATE RFPA RIGHT BY 1 BIT
0882      LD
0883      LD
0884      RRA
0885      LD
0886      LD
0887      RRA
0888      LD
0889      LD
088A      RRA
088B      LD
088C      DEC         ;DECREMENT THE BIT COUNTER
088D      LD          ;XFR TEST BYTE
088E      JR          ;LOOP IF ANOTHER BIT TO DO
0890  Z0890  POP         ;RESTORE THE FPA1 BYTE PTR
0891      RET         ;RET TO ANOTHER BYTE OR
                        ;NORMALIZE RESULT
0892  Z0892  LD          ;SHUFFLE 8 BITS RIGHT
0893      LD
0894      LD
0895      LD
0896      RET

;*****
;DIVISION, SINGLE PRECISION
;      0897 -> FPA1 / 10.0          -> FPA1
;      08A0 -> STACK VALUE / FPA1 -> FPA1
;      08A2 -> RFPA / FPA1         -> FPA1
;*****
0897  Z0897  CALL        ;FPA1 TO STACK
089A      LD          ;10.0 TO FPA1
089D      CALL
08A0  Z08A0  POP         ;RECOVER ORIG FPA1 INTO RFPA
08A1      POP
08A2  DIVSNG CALL        ;DIV BY ZER ERROR IF
08A5      JP         ;DIVISOR IS 0.0
08A8      LD          ;INIT TO SUB EXPONENTS
08AA      CALL        ;THEN DO IT
08AD      INC         ;BUILD VALUES INTO
08AE      INC         ;RAM ROUTINE 4080-408D
08AF      DEC         ;PT TO LOW ORDER DIVISOR

```

Math Routines: Disassembly

```

08B0      LD          ;BYTE, PICK IT UP & STUFF
08B1      LD          ;INTO RAM ROUTINE
08B4      DEC         ;PT TO MIDDLE DIVISOR
08B5      LD          ;BYTE, PICK IT UP & STUFF
08B6      LD          ;INTO RAM ROUTINE
08B9      DEC         ;PT TO HIGH ORDER DIVISOR
08BA      LD          ;BYTE, PICK IT UP & STUFF
08BB      LD          ;INTO RAM ROUTINE
08BE      LD          ;PLACE DIVIDEND MANTISSA
08BF      EX          ;INTO REGS B,H,& L
08C0      XOR         ;CLEAR THE RFPA MANTISSA
08C1      LD
08C2      LD
08C3      LD
08C4      LD          ;CLEAR "TEST" BYTE
08C7 Z08C7  PUSH      ;SAVE DIVIDEND
08C8      PUSH
08C9      LD          ;P/U DIVIDEND HIGH ORDER
08CA      CALL        ;RAM ROUTINE BUILT ABOVE

;*****
;
;   RAM ROUTINE AS FOLLOWS
;
;   4080H  SUB      N          ;SUB HIGH ORDER DIVISOR
;
;         LD      L,A
;
;         LD      A,H          ;P/U DIVIDEND MID ORDER
;
;   4084H  SBC      A,N          ;SUB MID ORDER DIVISOR
;
;         LD      H,A
;
;         LD      A,B          ;P/U DIVIDEND LOW ORDER
;
;   4088H  SBC      A,N          ;SUB LOW ORDER DIVISOR
;
;         LD      B,A
;
;   408BH  LD      A,N          ;SET TO TEST OVERFLOW
;
;         RET
;
;*****
;
;   SUBTRACT 0 FROM TEST BYTE IF HIGH ORDER DID NOT CARRY
;           1 FROM TEST BYTE IF HIGH ORDER DID CARRY
;
;   THEN SWITCH CARRY FLAG FROM THIS SUBTRACT
;
;*****
08CD      SBC
08CF      CCF          ;SWITCH RESULT OF CARRY
08D0      JR          ;BYPASS IF ORIGINALLY CARRIED
08D2      LD          ;RESET "TEST" VALUE
08D5      POP         ;POP OFF DIVIDEND
08D6      POP         ;AND IGNORE IT
08D7      SCF         ;RESET THE CARRY FLAG
08D8      DEFB      0D2H ;HIDE NEXT 2 INST WITH 'JP NC'
08D9 Z08D9  POP         ;POP DIVIDEND INTO B, H, & L
08DA      POP
08DB      LD          ;TEST FOR C HAVING BIT 7
08DC      INC         ;SET (I.E. SHOWS UP AS A
08DD      DEC         ;NEGATIVE VALUE)
08DE      RRA         ;P/U CARRY FLAG INTO BIT 7

```

Math Routines: Disassembly

```

;*****
;
; EXIT THIS PROCEDURE WHEN REG "C" OF RFPA
; HAS HAD A ONE SHIFTED INTO ITS BIT 7
;*****
08DF      JP          ;JUMP IF "C" HAD BIT 7 SET
08E2      RLA         ;ELSE RESTORE STATE OF CARRY
08E3      LD          ;NOW SHIFT THE REGISTER
08E4      RLA         ;FLOATING POINT ACCUMULATOR
08E5      LD          ;ONE BIT LEFT
08E6      LD          ;IF CARRY WAS SET FROM
08E7      RLA         ;08D7H, A BIT IS SHIFTED
08E8      LD          ;INTO THE RFPA
08E9      LD
08EA      RLA
08EB      LD
08EC      ADD         ;SHIFT DIVIDEND (B, H, L)
08ED      LD          ;ONE BIT LEFT
08EE      RLA
08EF      LD
08F0      LD          ;SHIFT "TEST" VALUE
08F3      RLA         ;ONE BIT LEFT
08F4      LD
08F7      LD          ;RECYCLE UNTIL RFPA IS
08F8      OR          ;COMPLETELY VOIDED
08F9      OR
08FA      JR
08FC      PUSH
08FD      LD          ;DEC RESULT EXPONENT SINCE
0900      DEC         ;DIVISOR WAS > DIVIDEND
0901      POP
0902      JR          ;RECYCLE IF NOT ZEROED
0904      JP          ;ELSE OVERFLOW ERROR

;*****
;ROUTINE TO PERFORM EXPONENT ADDITION OR SUBTRACTION
;*****
0907      Z0907      LD          ;INIT FOR SUBTRACTION
0909      DEFB        2EH      ;HIDE NEXT INST WITH 'LD L'
090A      Z090A      XOR         ;INIT FOR ADDITION
090B      LD
090E      LD
090F      INC
0910      XOR
0911      LD
0912      LD
0914      Z0914      LD          ;TEST EXPONENT FOR ZERO
0915      OR
0916      JR
0918      LD          ;L HAS 'FF' FROM DIV; '00' FROM MUL
0919      LD
091C      XOR

```

Math Routines: Disassembly

```

091D      ADD
091E      LD
091F      RRA          ;BRING CARRY INTO BIT 7
0920      XOR
0921      LD          ;P/U THE NEW EXPONENT
0922      JP
0925      ADD
0927      LD
0928      JP          ;-> POP HL, RET
092B      CALL        ;SWITCH SIGN BIT
092E      LD
092F      Z092F      DEC
0930      RET
0931      Z0931      CALL
0934      CPL
0935      POP
0936      Z0936      OR
0937      Z0937      POP
0938      JP          ;ZERO FPA1
093B      JP
;*****
;MULTIPLY FPA1 BY 10.0
;*****
093E      Z093E      CALL        ;FPA1 -> RFPA
0941      LD          ;RET IF VALUE = 0.0
0942      OR
0943      RET
0944      ADD          ;MULT BY 4
0946      JP
0949      LD
094A      CALL        ;ADD ONCE TO MULT BY 5
094D      LD          ;INC FPA1 EXPONENT
0950      INC          ;MULT RESULT BY 2 = X 10
0951      RET          ;RETURN IF IT DID NOT GO
0952      JP          ;FROM FF TO 00, ELSE ERROR
;*****
;ROUTINE CHECKS FPA1 FOR MINUS, ZERO, PLUS
; RETURNS -1, 0, +1
;*****
0955      CKRMZP      LD          ;RETURN IF EXPONENT = 0
0958      OR
0959      RET
095A      LD          ;GET SIGN BIT
095D      DEFB        OFEH      ;HIDE NEXT INST WITH 'CP'
095E      Z095E      CPL
095F      Z095F      RLA          ;SIGN BIT -> CARRY FLAG
0960      Z0960      SBC          ;CONVERT TO -1 OR +1
0961      RET
0962      INC
0963      RET

```

Math Routines: Disassembly

```

;*****
;INIT RFPA WITH 128.0
;*****
0964 Z0964 LD ;INIT RFPA WITH 128.0
0966 LD
0969 Z0969 LD
096C LD
096D LD
096E LD
0970 INC
0971 LD
0973 RLA
0974 JP

;*****
;INITIAL PROCESSING OF ABS(X)
;*****
0977 ABS CALL ;TEST FOR +, 0, -
097A RET ;RETURN IF + OR 0
097B Z097B RST ;CHECK TYPE
097C JP
097F JP
0982 Z0982 LD ;CPL SIGN BIT OF FPA1
0985 LD
0986 XOR
0988 LD
0989 RET

;*****
;INITIAL PROCESSING OF SGN(X)
; RETURN VALUE (-1, 0, +1) IN INTEGER ACCUM
;*****
098A SGN CALL ;TEST VALUE
098D XPNDCF LD ;PLACE -1, 0, OR +1 -> L
098E RLA ;ZERO OUT IF ZERO OR +
098F SBC ;MAKE FF IF MINUS
0990 LD ;THEN LOAD INTO H
0991 JP ;CHANGE TYPE TO INT & HL -> ACCUM

;*****
;VALUE TESTING FOR ABS & SGN
;*****
0994 Z0994 RST
0995 JP
0998 JP ;JUMP IF SGL OR DBL
099B LD ;GET INTEGER
099E LD ;RET IF ZERO
099F OR
09A0 RET
09A1 LD ;HIGH ORDER BYTE TO A
09A2 JR

;*****
;TRANSFER FPA1 TO STACK

```

Math Routines: Disassembly

```

;*****
09A4 STKFP1  EX
09A5      LD
09A8      EX
09A9      PUSH
09AA      LD
09AD      EX
09AE      PUSH
09AF      EX
09B0      RET

;*****
;TRANSFERS SNGL POINTED TO BY HL INTO FPA1
;*****
09B1 HL FPA1  CALL
09B4 SNGFPA  EX
09B5      LD
09B8      LD
09B9      LD
09BA      LD
09BD      EX
09BE      RET

;*****
;TRANSFER FPA1 (OR VALUE POINTED TO BY HL) INTO REGS RFPA
;*****
09BF LD FPA1  LD          ;LD FPA1 -> EDCB REGS
09C2 LD FPHL  LD          ;LD (HL) -> EDCB REGS
09C3      INC
09C4      LD
09C5      INC
09C6      LD
09C7      INC
09C8      LD
09C9 Z09C9  INC
09CA      RET

;*****
;TRANSFER FPA1 TO MEMORY POINTED TO BY HL
;*****
09CB FPAMEM  LD          ;TRANSFER FPA1 TO MEMORY
09CE      LD          ;POINTED TO BY HL
09D0      JR

;*****
;TRANSFERS DATA VALUE FROM (DE) TO (HL) DEPENDING ON TYPE
;OR FROM (HL) TO (DE) DEPENDING ON ENTRY POINT
;*****
09D2 Z09D2  EX          ;TRANSFER 'TYPE' BYTES
09D3 MOVDAT  LD          ;FROM (HL) -> (DE)
09D6      LD
09D7 Z09D7  LD
09D8      LD
09D9      INC

```

Math Routines: Disassembly

```

09DA          INC
09DB          DEC
09DC          JR
09DE          RET

;*****
;ALTER SIGN BIT OF FLOATING POINT VALUES
;RESULTANT SIGN BIT IN A IS 1 IF BOTH + OR BOTH -
;          SIGN BIT          IS 0 IF BOTH UNEQUAL SIGN
;*****
09DF  Z09DF   LD          ;PT TO MSB
09E2          LD          ;SET THE SIGN BIT
09E3          RLCA
09E4          SCF
09E5          RRA
09E6          LD          ;& REPL MSB
09E7          CCF
09E8          RRA
09E9          INC          ;PLACE ORIG MSB BUT WITH A
09EA          INC          ;COMPLEMENTED SIGN BIT
09EB          LD          ;INTO ADDR 4125H
09EC          LD          ;SET SIGN BIT OF VAR2
09ED          RLCA
09EE          SCF
09EF          RRA
09F0          LD          ;& REPL MSB
09F1          RRA
09F2          XOR
09F3          RET

;*****
;VARIOUS DATA TRANSFERS FROM VARIABLE TABLES TO ACCUMULATOR
;*****
09F4  Z09F4   LD
09F7  Z09F7   LD
09FA          JR

```

Math Routines: Disassembly

```

;*****
;SINGLE PRECISION COMPARISONS
;      -1 IF FPA1 < RFPA
;      0 IF FPA1 = RFPA
;      +1 IF FPA1 > RFPA
;*****
0A0C CPRSNG LD ;IF RFPA IS ZERO, THEN RESULT
0A0D OR ;IS BASED ON FPA1'S SIGN
0A0E JP
0A11 LD ;ELSE STACK RET TO -1/+1 RTN
0A14 PUSH ;WHICH WILL SET THE RESULT
0A15 CALL ;TEST SIGN OF FPA1
0A18 LD ;LOAD SIGN BYTE OF RFPA
0A19 RET ;IF FPA1 IS ZERO, RESULT IS
0A1A LD ;BASED ON RFPA'S SIGN, ELSE
0A1D XOR ;TEST IF EXACTLY ONE VAR IS
0A1E LD ;NEGATIVE (VIA EXCL OR)
0A1F RET ;IF SO, THEN RESULT BASED ON RFPA
0A20 CALL ;ELSE COMPARE EACH BYTE
0A23 Z0A23 RRA ;WILL RET HERE IF UNEQUAL
0A24 XOR ;XOR C-FLG OF RESULT WITH
0A25 RET ;RFPA'S SIGN & EXIT
0A26 Z0A26 INC ;COMPARE EACH BYTE OF FPA1
0A27 LD ;WITH EACH BYTE OF RFPA
0A28 CP ;CPR EXPONENTS
0A29 RET
0A2A DEC
0A2B LD ;CPR HI ORDER MANTISSA
0A2C CP
0A2D RET
0A2E DEC
0A2F LD ;CPR MID MANTISSA
0A30 CP
0A31 RET
0A32 DEC
0A33 LD ;CPR LOW ORDER MANTISSA
0A34 SUB
0A35 RET
0A36 POP ;IF NONE DIFFER, THEY ARE EQUAL!
0A37 POP ;POP OFF THE RET CODES AS
0A38 RET ;RESULT IS CORRECTLY ZERO
;*****
;INTEGER COMPARISONS
;*****
0A39 CPRINT LD ;TEST IF ONE VAR IS NEGATIVE
0A3A XOR ;BY EXCL OR THE SIGNS
0A3B LD ;INIT TO CPR ON HL ONLY
0A3C JP ;IF ONE NEG, THEN RESULT BASED
0A3F CP ;ON SIGN OF HL, ELSE CPR

```

Math Routines: Disassembly

```

0A40          JP          ;HI ORDER, THEN LO ORDER
0A43          LD          ;IF NECESSARY
0A44          SUB
0A45          JP
0A48          RET

;*****
;DOUBLE PRECISION COMPARISONS
;*****

0A49          LD
0A4C          CALL
0A4F Z0A4F    LD          ;TEST FPA2 FOR ZERO
0A52          LD
0A53          OR          ;IF FPA2 = ZERO, THEN RESULT
0A54          JP          ;BASED ON SIGN OF FPA1
0A57          LD          ;STACK RET TO +1/-1 RTN
0A5A          PUSH
0A5B          CALL        ;TEST SIGN OF FPA1
0A5E          DEC        ;PT TO SIGN BYTE OF FPA2
0A5F          LD          ;& PUT IT INTO REG C
0A60          LD          ;IF FPA1 IS ZERO, THEN RESULT
0A61          RET        ;BASED ON FPA2'S SIGN
0A62          LD          ;ELSE EXCL OR THE SIGNS TO
0A65          XOR        ;SEE IF ONE VAR IS NEGATIVE
0A66          LD          ;IF EXACTLY ONE IS NEGATIVE
0A67          RET        ;RESULT BASED ON FPA2'S SIGN
0A68          INC        ;ELSE POINT DE & HL TO THE
0A69          INC        ;EXPONENT BYTE TO CPR 8 BYTES
0A6A          LD          ;INIT LOOP CTR FOR 8 BYTES
0A6C Z0A6C    LD          ;COMPARE EACH BYTE IN TURN
0A6D          SUB
0A6E          JP          ;EXIT IF UNEQUAL
0A71          DEC        ;ELSE DEC PTRS
0A72          DEC
0A73          DEC        ;& DEC THE LOOP COUNTER
0A74          JR          ;CYCLE IF MORE
0A76          POP
0A77          RET        ;ALL ARE SAME, REMOVE RET TO
0A78 CPRDBL   CALL        ;095EH & EXIT AS RESULT IS ZERO
0A7B          JP          ;INIT CALL TO CPR. IF UNEQUAL,
0A7E          RET        ;CVRT RESULT TO -1 OR +1
                     ;ELSE RET WITH ZERO

;*****
;PROCESS CINT(X)
;*****

0A7F CINT     RST          ;CK TYPE
0A80          LD
0A83          RET        ;RETURN IF INTEGER
0A84          JP
0A87          CALL        ;CVRT DBL TO SNGL
0A8A          LD          ;ESTABLISH RETURN IN CASE
0A8D          PUSH

```

Math Routines: Disassembly

```

0A8E Z0A8E LD
0A91 CP ;CK IF > 32768
0A93 JR
0A95 CALL
0A98 EX
0A99 Z0A99 POP
0A9A SAVINT LD
0A9D LD ;CHG TYPE TO INTEGER
0A9F SETINT LD
0AA2 RET
0AA3 Z0AA3 LD ;-32768
0AA6 LD
0AA9 CALL
0AAC RET
0AAD LD
0AAE LD
0AAF JR
;*****
;PROCESS CSNG(X)
;*****
0AB1 CSNG RST
0AB2 RET ;RETURN IF SNGL
0AB3 JP ;JUMP IF INTEGER
0AB6 JP ;ERR IF NOT DBL
0AB9 DBLSNG CALL ;CVRT DBL TO SNGL FIRST
0ABC CALL
0ABF LD ;TEST FOR ZERO
0AC0 OR
0AC1 RET
0AC2 CALL
0AC5 LD
0AC8 LD
0AC9 JP
0ACC Z0ACC LD ;CVRT INTEGER TO SNGL
0ACF Z0ACF CALL ;CHG TYPE TO SNGL
0AD2 LD
0AD3 LD
0AD4 LD
0AD6 LD ;32768
0AD8 JP
;*****
;PROCESS CDBL(X)
;*****
0ADB CDBL RST ;RET IF DBL
0ADC RET
0ADD JP ;IF STRING
0AE0 CALL ;CALL IF INT
0AE3 Z0AE3 LD ;ZERO THE EXTENDED PART OF FPA1
0AE6 LD
0AE9 LD

```

Math Routines: Disassembly

```

0AEC  SETDBL  LD          ;CHANGE TYPE TO DBL
0AEE  DEFEB   1          ;HIDE NEXT INST WITH 'LD BC'
0AEF  SETSNG  LD          ;CHANGE TYPE TO SNGL
0AF1  JP
;*****
;CHECK TYPE OF CURRENT VARIABLE & PROVIDE TM ERROR IF STRING
;*****
0AF4  CHKSTR  RST          ;ROUTINE CHECKS CURRENT VARIABLE
0AF5  RET          ;FOR STRING, RET IF OK ELSE ERROR
0AF6  TMERR   LD          ;TYPE MISMATCH ERROR
0AF8  JP
;*****
;LD B,C,D,E WITH REG. A
;*****
0AFB  Z0AFB   LD
0AFC          LD
0AFD          LD
0AFE          LD
0AFF          OR
0B00          RET          ;RET IF ZERO RFPA
0B01  PUSH
0B02  CALL          ;FPA1 TO RFPA
0B05  CALL
0B08  XOR
0B09  LD
0B0A  CALL          ;REDUCE BCDE BY ONE
0B0D  LD
0B0F  SUB
0B10  CALL          ;SHIFT RIGHT "A" BITS
0B13  LD
0B14  RLA
0B15  CALL          ;IF H NEG, THEN ADD 1 TO RFPA
0B18  LD
0B1A  CALL          ;TWO'S COMP THE RFPA
0B1D  POP
0B1E  RET
0B1F  Z0B1F   DEC          ;REDUCE "BCDE" BY 1
0B20          LD          ;TEST IF DE WENT FROM
0B21          AND          ;0000 TO FFFF
0B22          INC          ;IF SO, THIS IS 0
0B23          RET          ;RET IF NOT SO
0B24          DEC          ;ELSE DEC BC FOR CARRY
0B25          RET
;*****
;PROCESS FIX(X)
;*****
0B26  FIX     RST          ;RET IF INT
0B27          RET
0B28          CALL
0B2B          JP

```

Math Routines: Disassembly

```

0B2E          CALL          ;CHG SIGN BIT FM M TO P
0B31          CALL
0B34          JP            ;COMPLEMENT THE INTEGER
;*****
;PROCESS INT(X)
;*****
0B37 INT      RST
0B38          RET            ;RETURN IF INTEGER
0B39          JR            ;JUMP IF DOUBLE
0B3B          JR            ;IF STRING
;*****
;FIND INTEGER PART OF SINGLE PRECISION
;      ENTER AT 0B40H TO TAKE INT(FPA1)
;*****
0B3D          CALL          ;FIND CINT OF SNGL
0B40 Z0B40    LD
0B43          LD
0B44          CP
0B46          LD
0B49          RET
0B4A          LD            ;LD FPA1 EXP. INTO A
0B4B          CALL          ;PUT EXP. INTO B,C,D, & E
0B4E          LD
0B50          LD
0B51          PUSH
0B52          LD
0B53          RLA
0B54          CALL          ;TWO'S COMP RFPA THEN NORMALIZE
0B57          POP
0B58          RET
;*****
;FIND INTEGER PART OF DOUBLE PRECISION
;*****
0B59 INTDBL   LD            ;LD HL WITH ADDR OF FPA1 EXP
0B5C          LD            ;LD EXP TO ACCUM
0B5D          CP            ;VAL WITHIN RANGE OF INTEGER, CINT
0B5F          JP
0B62          JR
0B64          LD
0B65          DEC
0B66          LD            ;LD FIRST BYTE MANTISSA
0B67          XOR            ;FLIP SIGN BIT
0B69          LD            ;SET-UP FOR 6-BYTES OF MANTISSA
0B6B Z0B6B    DEC
0B6C          OR
0B6D          DEC
0B6E          JR            ;LOOP UNTIL DONE
0B70          OR
0B71          LD
0B74          JP

```

Math Routines: Disassembly

```

0B77      LD
0B78  Z0B78  CP
0B7A      RET
0B7B  Z0B7B  PUSH
0B7C      CALL      ;FPA1 -> RFPA
0B7F      CALL
0B82      XOR
0B83      DEC
0B84      LD
0B86      PUSH
0B87      CALL
0B8A      LD
0B8D      LD
0B8F      SUB
0B90      CALL      ;SHIFT RIGHT "A" BITS
0B93      POP
0B94      CALL      ;INC DBL BY ONE
0B97      XOR
0B98      LD
0B9B      POP
0B9C      RET
0B9D      JP      ;TWO'S COMPLEMENT
0BA0  Z0BA0  LD
0BA3  Z0BA3  LD
0BA4      DEC
0BA5      OR
0BA6      INC
0BA7      JR
0BA9      RET
;*****
;CALCULATION OF NUMBER OF BYTES USED IN A DIMENSION
;MULTIPLIES DE (TYPE LENGTH) BY BC (DIM) RESULT IN DE
;*****
0BAA      PUSH
0BAB      LD
0BAE      LD
0BAF      OR
0BB0      JR      ;JUMP IF DIM IS ZERO
0BB2      LD      ;ESTAB LOOP LIMIT
0BB4  Z0BB4  ADD
0BB5      JP      ;OVERFLOW ERROR
0BB8      EX      ;ADD DE+DE -> DE
0BB9      ADD
0BBA      EX
0BBB      JR
0BBD      ADD
0BBE      JP      ;OVERFLOW ERROR
0BC1  Z0BC1  DEC
0BC2      JR
0BC4  Z0BC4  EX

```

Math Routines: Disassembly

```

0BC5          POP
0BC6          RET
;*****
;INTEGER SUBTRACTION
;*****
0BC7 SUBINT LD          ;OBTAIN SIGN BIT &
0BC8          RLA        ;PLACE IN REG B
0BC9          SBC
0BCA          LD
0BCB          CALL        ;HL -> FPA1
0BCE          LD          ;ZERO REG A
0BCF          SBC
0BD0          JR          ;NOW ADD
;*****
;INTEGER ADDITION
;*****
0BD2 ADDINT LD          ;OBTAIN SIGN BIT &
0BD3          RLA        ;PLACE IN REG B
0BD4          SBC
0BD5 Z0BD5 LD
0BD6          PUSH        ;SAVE VALUE
0BD7          LD          ;SIGN BIT -> REG A
0BD8          RLA
0BD9          SBC
0BDA          ADD          ;ADD VAL2 TO VAL1
0BDB          ADC
0BDC          RRCA
0BDD          XOR
0BDE          JP          ;RESULT TO FPA1, TYPFLG=2
;*****
;ADDITION OVERFLOWED INTEGER LIMITS, CONVERT TO SINGLE
;*****
0BE1          PUSH
0BE2          EX          ;VAL2 -> HL
0BE3          CALL        ;CVRT VAL2 TO SNGL
0BE6          POP
0BE7          POP          ;RCVR VAL1
0BE8          CALL
0BEB          EX
0BEC          CALL
0BEF          JP          ;STACK -> RFPA -> ADDSNG
;*****
;INTEGER MULTIPLICATION
;*****
0BF2 MULINT LD          ;TEST FOR ZERO VALUE
0BF3          OR
0BF4          JP          ;HL -> FPA1, TYPFLG -> 2
0BF7          PUSH        ;SAVE VAL1
0BF8          PUSH        ;SAVE VAL2
0BF9          CALL        ;MAKE SURE BOTH VALS ARE +

```

Math Routines: Disassembly

0BFC		PUSH		;RESULT SIGN SAVED IN B
0BFD		LD		;NOW MULTIPLY HL BY DE
0BFE		LD		; RESULT IN HL (BOTH VALS
0BFF		LD		;ARE POSITIVE). INIT TO 0
0C02		LD		;INIT FOR 16 BITS
0C04	Z0C04	ADD		;SHIFT MULTIPLICAND
0C05		JR		;TEST FOR OVERFLOW
0C07		EX		;SHIFT MULTIPLIER 1 BIT LEFT
0C08		ADD		;BY ADDING IT TO ITSELF.
0C09		EX		;IF A 1-BIT IS NOT SHIFTED INTO
0C0A		JR		;THE CARRY, RECYCLE, ELSE
0C0C		ADD		;ADD IN AN 'HL' THEN
0C0D		JP		;TEST FOR OVERFLOW
0C10	Z0C10	DEC		;REDUCE BIT COUNTER
0C11		JR		;LOOP IF MORE TO DO
0C13		POP		;RESTORE REGS
0C14		POP		
0C15		LD		;TEST RESULT FOR OVERFLOW
0C16		OR		;INTO NEGATIVE (BIT 7 SET)
0C17		JP		;OVERFLOW IF BIT 7 SET
0C1A		POP		;ELSE RESTORE REG
0C1B		LD		;PUT SIGN BACK INTO REG A
0C1C		JP		;& EXIT
0C1F	Z0C1F	XOR		;TURN OFF SIGN BIT OF NEG RESULT
0C21		OR		
0C22		JR		
0C24		EX		
0C25		DEFB	1	;HIDE NEXT 2 INST WITH 'LD BC'
0C26	Z0C26	POP		;MULTIPLICAND OVERFLOWED
0C27		POP		
0C28		CALL		;CVRT VAL TO SNGL
0C2B		POP		
0C2C		CALL		;& STACK IT AWAY
0C2F		CALL		;CVRT OTHER VAL TO SNGL
0C32	Z0C32	POP		;RESTORE STACKED VAL
0C33		POP		
0C34		JP		;NOW MULTIPLY SINGLE
0C37	Z0C37	LD		
0C38		OR		
0C39		POP		
0C3A		JP		;CHG TYPE TO INT & HL -> ACCUM
0C3D		PUSH		
0C3E		CALL		;CVRT VAL TO SNGL
0C41		POP		
0C42		JP		
0C45	Z0C45	LD		;EXCL OR SIGN BITS
0C46		XOR		;& SAVE IN REG B
0C47		LD		
0C48		CALL		;MAKE VAL IN HL POSITIVE IF NOT
0C4B		EX		;MAKE VAL IN DE POSITIVE IF NOT

Math Routines: Disassembly

```

0C4C Z0C4C LD ;IF THE VAL IS POSITIVE,
0C4D Z0C4D OR ;SAVE IT IN FPA1
0C4E JP ;AS AN INTEGER RESULT
0C51 Z0C51 XOR ;ELSE CVRT IT TO POSITIVE
0C52 LD ;BY TAKING THE TWO'S
0C53 SUB ;COMPLEMENT OF HL
0C54 LD
0C55 LD
0C56 SBC
0C57 LD
0C58 JP ;THEN SAVING THE INTEGER RESULT
;*****
;PROCESS ABSOLUTE VALUE OF AN INTEGER
;*****
0C5B ABSINT LD ;P/U INTEGER
0C5E CALL ;COMPLEMENT IT
0C61 LD
0C62 XOR
0C64 OR
0C65 RET
0C66 EX
0C67 CALL ;CHG TYPE TO SNGL
0C6A XOR
0C6B Z0C6B LD
0C6D JP
;*****
;DOUBLE PRECISION SUBTRACTION
;*****
0C70 SUBDBL LD ;CHANGE SIGN OF FPA2
0C73 LD
0C74 XOR ;CHANGE SIGN, THEN ADD
0C76 LD
;*****
; DOUBLE PRECISION ADDITION
;*****
0C77 ADDDBL LD
0C7A LD ;RETURN IF FPA2 IS ZERO
0C7B OR ;AS FPA1 WOULD HAVE THE
0C7C RET ;ANSWER
0C7D LD ;SAVE THE EXPONENT IN B
0C7E DEC
0C7F LD ;P/U HIGH BYTE (CONTAINS SIGN)
0C80 LD ;IF FPA1 IS ZERO, THEN
0C83 LD ;SWAP FPA2 WITH FPA1 AS
0C84 OR ;FPA2 WOULD BE THE ANSWER
0C85 JP
0C88 SUB ;GET DIFF IN EXPONENTS
0C89 JR ;JUMP IF FPA2 < FPA1
0C8B CPL ;CVRT EXPON DIFF TO +
0C8C INC

```

Math Routines: Disassembly

```

;*****
;EXCHANGE FPA1 & FPA2
;*****
0C8D          PUSH
0C8E          LD          ;SET LOOP FOR 8 BYTES
0C90          INC
0C91          PUSH
0C92  AD1      LD
0C93          LD
0C94          LD
0C95          LD
0C96          LD
0C97          DEC
0C98          DEC
0C99          DEC          ;DEC CTR & CYCLE IF MORE
0C9A          JR
0C9C          POP          ;RECOVER 412E
0C9D          LD          ;EXPON -> B
0C9E          DEC
0C9F          LD          ;SIGN BYTE -> C
OCA0          POP
;*****
;DON'T ADD IF THE DIFFERENCE BETWEEN VALUES > 10**17
;*****
OCA1  ZOCA1    CP          ;2**57 APPX 10**17
OCA3          RET
OCA4          PUSH
OCA5          CALL          ;TURN ON SIGN BITS
OCA8          INC          ;PT TO 4126H & ZERO IT
OCA9          LD
OCAB          LD
OCAC          POP
OCAD          LD
OCB0          CALL          ;DIV VAR1 MANTISSA BY THE
OCB3          LD          ;DIFF IN EXPONENTS
OCB6          LD
OCB9          LD          ;TEST RESULT OF SIGNS
OCBA          OR
OCBB          JP          ;JUMP IF SIGN BITS WERE <>
OCBE          CALL          ;ADD DBL (HL) TO (DE)
OCC1          JP          ;INC EXPON IF ADD OVERFLOWED
OCC4          EX          ;PT TO EXPON AND ADD 1 DUE
OCC5          INC          ;TO CARRY ON MANTISSA ADD
OCC6          JP          ;ERROR IF EXPON ADD -> 0
OCC9          CALL          ;DIVIDE MANTISSA BY 2 FOR
OCCC          JP          ;EXPONENT INCREASE
;*****
; SUBTRACT ONE NUMBER FROM ANOTHER WHEN 'SIGNS' ARE <>
;*****
OCCF  ZOCCF    CALL          ;SUB DBL (HL) FROM (DE)

```

Math Routines: Disassembly

```

OCD2          LD
OCD5          CALL          ;TWO'S COMP FPA1
OCD8  ZOCD8   XOR          ;SET SHIFT CTR TO ZERO
OCD9  DSHFT8  LD          ;SET SHIFT CTR FROM ACCUM
OCDA          LD          ;IF HIGH BYTE IS ZERO,
OCDD          OR          ;SHIFT LEFT 8 BITS
OCDE          JR          ;ELSE BYPASS 8-BIT SHIFTER
OCE0          LD          ;PT TO LOW-1
OCE3          LD          ;INIT BYTE COUNTER
OCE5  DLOOP8  LD          ;SHIFT FPA1 DBL LEFT 8 BITS
OCE6          LD
OCE7          LD
OCE8          INC
OCE9          DEC          ;DEC THE BYTE CTR
OCEA          JR
OCEC          LD          ;P/U THS SHIFT CTR
OCED          SUB          ;& COUNT IT DOWN BY 8
OCEF          CP          ;IF MANTISSA WAS NOT SHIFTED
OCF1          JR          ;OUT OF FPA1, THEN CYCLE
OCF3          JP          ;ELSE ZERO FPA1 & GO HOME
;*****
; DBL PREC SINGLE BIT SHIFTER (LEFT)
;*****
OCF6  DSHFT1  DEC
OCF7          LD          ;SHIFT 1 BIT LEFT UNTIL
OCFA          CALL          ;THE SIGN BIT GOES TO 1
OCFD          OR
OCFE  DCHKP   JP
;*****
; A 1 HAS SHIFTED INTO THE SIGN POSITION
;*****
OD01          LD          ;P/U THE SHIFT COUNTER
OD02          OR          ;SEE IF SHIFTING HAD OCCURED
OD03          JR
;*****
; THE NBR HAD TO BE SHIFTED TO BE NORMALIZED.
; CORRECT THE EXPONENT.
;*****
OD05          LD          ;ADD THE SHIFT COUNTER
OD08          ADD          ;WHICH IS NEGATIVE, TO
OD09          LD          ;FPA1'S EXPONENT
OD0A          JP          ;ZERO FPA1 IF SHIFT < EXPON
OD0D          RET          ;RET WITH FPA1=0 IF SHIFT=EXPON
;*****
; SHIFT WAS GREATER THAN EXPONENT
;*****
OD0E  ZOD0E   LD
OD11  ZOD11   OR          ;IS BIT 7 SET?
OD12          CALL          ;INC FPA1 BY 1 IF IT IS
OD15          LD

```

Math Routines: Disassembly

```

0D18          LD          ;P/U THE PROCESSED SIGN BITS
0D19          AND          ;& STRIP OFF ALL BUT THE SIGN
0D1B          DEC
0D1C          DEC          ;PT TO FPA1'S SIGN BYTE
0D1D          XOR          ;& INSERT IN THE CORRECT SIGN
0D1E          LD
0D1F          RET

;*****
; INCREMENT DBL PREC FPA1 BY 1
;*****
0D20  INCDBL   LD
0D23          LD          ;SET CTR FOR MANTISSA BYTES
0D25  DLOOP7  INC          ;INC THE BYTE
0D26          RET          ;RET IF NO 'CARRY'
0D27          INC          ;ELSE PT TO NEXT BYTE
0D28          DEC          ;DEC THE BYTE CTR & CYCLE IF MORE
0D29          JR
0D2B          INC          ;INC THE EXPONENT & ERROR
0D2C          JP          ;IF IT OVERFLOWED
0D2F          DEC          ;PUT THE 'SIGN' BIT 1 BACK
0D30          LD
0D32          RET

;*****
; ADD DBL PREC MANTISSA HL TO DE
;*****
0D33  DAF1F2  LD          ;ADD FPA2 TO FPA1
0D36  DAF1HL  LD          ;ADD (HL) TO FPA1
0D39  DADEHL  LD          ;ADD (HL) TO (DE)
0D3B          XOR
0D3C  DA1     LD
0D3D          ADC
0D3E          LD
0D3F          INC
0D40          INC
0D41          DEC
0D42          JR
0D44          RET

;*****
; SUBTRACT DBL PREC MANTISSA HL FROM DE
;*****
0D45  DSF1F2  LD          ;SUB FPA2 FM FPA1
0D48  DSF1HL  LD          ;SUB (HL) FM FPA1
0D4B  DSDEHL  LD          ;SUB (HL) FM (DE)
0D4D          XOR
0D4E  DS1     LD
0D4F          SBC
0D50          LD
0D51          INC
0D52          INC
0D53          DEC

```

Math Routines: Disassembly

```

0D54          JR
0D56          RET
;*****
; TWO'S COMPLEMENT THE DBL PREC FPA1
;*****
0D57 D2SCMP   LD          ;INVERT THE RESULT SIGN BIT
0D58          CPL
0D59          LD
0D5A          LD
0D5D          LD          ;INIT BYTE COUNTER
0D5F          XOR
0D60 Z0D60   LD          ;SET REG C
0D61 Z0D61   LD          ;REFRESH ACCUM TO ZERO
0D62          SBC          ;COMPLEMENT A BYTE
0D63          LD
0D64          INC          ;PT TO NEXT BYTE
0D65          DEC          ;DEC BYTE CTR & CYCLE IF MORE
0D66          JR
0D68          RET
;*****
; PERFORM A RIGHT CIRCULAR SHIFT OF A MANTISSA BASED ON REG-A
;*****
0D69 DSHTR    LD
0D6A          PUSH
0D6B DSHTR8   SUB          ;TEST FOR 8 OR MORE BITS
0D6D          JR          ;TO SHIFT
0D6F          POP
0D70 Z0D70   PUSH
0D71          LD          ;INIT D TO 8, E TO 0
0D74 Z0D74   LD
0D75          LD          ;JUGGLE THE 8 BYTES
0D76          LD
0D77          DEC
0D78          DEC          ;DEC BYTE CTR
0D79          JR          ;CYCLE IF MORE BYTES
0D7B          JR
;*****
; DBL PREC SINGLE BIT RIGHT SHIFTER
;*****
0D7D Z0D7D   ADD          ;ADD BACK THE 8 + 1
0D7F          LD          ;INIT THE SHIFT COUNTER
0D80 DSHTRS   XOR          ;ZERO A & REDUCE CTR
0D81          POP
0D82          DEC
0D83          RET          ;FINISHED WHEN CTR RUNS OUT
;*****
; DBL PREC SHIFT RIGHT 'D' BITS
;*****
0D84 DSHTRD   PUSH          ;SAVE THE POINTER
0D85          LD          ;INIT FOR 8 BYTES

```

Math Routines: Disassembly

```

0D87 Z0D87 LD ;SHIFT EACH BYTE ONE BIT
0D88 RRA
0D89 LD
0D8A DEC ;PT TO NEXT LOWER BYTE
0D8B DEC ;DEC THE COUNTER
0D8C JR ;UNTIL 8 BYTES DONE
0D8E JR
;*****
; DBL PREC SINGLE BIT SHIFT RIGHT (8-BYTES)
;*****
0D90 DSHTR1 LD
0D93 LD ;INIT FOR 1 BIT
0D95 JR
;*****
;DOUBLE PRECISION MULTIPLICATION
;*****
0DA1 MULDBL CALL ;CHECK -,0,+ AND RETURN IF ZERO
0DA4 RET
0DA5 CALL ;ADD THE EXPONENTS

0DA8 CALL
0DAB LD
0DAC INC
0DAD LD ; INIT FOR 7 BYTE MANTISSA
0DAF Z0DAF LD ; PICKUP TEST BYTE AND POINT TO NEXT
0DB0 INC
0DB1 OR
0DB2 PUSH
0DB3 JR ; BYPASS BIT CHECK IF BYTE IS ZERO
0DB5 LD ;SET FOR 8 TIMES
0DB7 Z0DB7 PUSH ;SAVE COUNTER REGISTERS
0DB8 RRA
0DB9 LD ;SAVE ACCUM
0DBA CALL ;ADD MANTISSA 2 TO MANTISSA 1
0DBD CALL ;DBL SINGLE BIT SHIFT RIGHT (8 BYTES)
0DC0 LD ;RESTORE ACCUM
0DC1 POP ;RESTORE LOOP REGISTERS
0DC2 DEC ;DEC INNER LOOP REGISTER
0DC3 JR ;LOOP IF NOT DONE
0DC5 Z0DC5 POP ;RECOVER POINTER TO BYTE
0DC6 DEC ;OUTER LOOP REGISTER
0DC7 JR ;LOOP IF NOT DONE
0DC9 JP ;NORMALIZE
0DCC Z0DCC LD ;LD FPA1 MANTISSA
0DCF CALL ;RIGHT CIRCULAR BYTE SHIFT
0DD2 JR ;CONTINUE LOOP
;*****
;FLOATING POINT DATA
;*****

```

Math Routines: Disassembly

```

0DD4      DEFW    0000      ;10.0 (DOUBLE PRECISION)
0DD6      DEFW    0000
0DD8      DEFW    0000      ;10.0 (SINGLE PRECISION)
0DDA      DEFW    8420H

;*****
;DOUBLE PRECISION DIVISION ROUTINE
;*****
0DDC      ZODDC      LD          ;LD 10.0D
0DDF      LD          ;LD ADDRESS OF FPA1
0DE2      CALL       ;MOVE 10.0D TO FPA2
0DE5      DIVDBL     LD          ;LOAD FPA2 AND
0DE8      OR         ;TEST FOR ZERO
0DE9      JP         ;DIV BY ZERO ERR IF ZERO
0DEC      CALL       ;EXPONENT SUBTRACTION
0DEF      INC
0DF0      INC
0DF1      CALL       ;MOVE FPA2 TO DIVISION WORK AREA
0DF4      LD
0DF7      LD
0DF8      LD
0DF9      ZODF9      LD
0DFC      LD
0DFF      CALL       ;SUBTRACT DBL MANTISSA (HL) FROM (DE)
0E02      LD
0E03      SBC
0E04      CCF
0E05      JR
0E07      LD
0E0A      LD
0E0D      CALL       ;ADD (HL) TO (DE)
0E10      XOR
0E11      DEFB      0DAH      ;HIDE NEXT TWO INSTRUCTIONS W/JP C,0412H
0E12      ZOE12      LD
0E13      INC
0E14      LD          ;LD ACCUM 1ST BYTE FPA1 MANTISSA
0E17      INC          ;TEST ACCUM
0E18      DEC
0E19      RRA
0E1A      JP         ;JP IF ACCUM NEGATIVE
0E1D      RLA
0E1E      LD
0E21      LD          ;SET UP FOR 7 LOOPS
0E23      CALL       ;DBL PREC. SINGLE BIT SHIFT LEFT (7 BYTES)
0E26      LD          ;POINT TO DIVISION WORK AREA
0E29      CALL       ;DBL PREC. SINGLE BIT SHIFT LEFT (8)
0E2C      LD
0E2D      OR
0E2E      JR
0E30      LD          ;PT TO FPA1 EXP.
0E33      DEC          ;DEC EXPONENT

```

Math Routines: Disassembly

```

0E34          JR
0E36          JP          ;JP OVERFLOW ERROR
0E39  Z0E39   LD
0E3A          LD
0E3D          DEC
0E3E          LD
0E41          LD
0E44  Z0E44   LD          ;MOVE FPA1 TO DIVISION WORK AREA
0E45          LD
0E46          LD
0E47          DEC
0E48          DEC
0E49          DEC
0E4A          JR
0E4C          RET

;*****
; MULTIPLY A DOUBLE PRECISION VALUE BY 10.0
;*****
0E4D  Z0E4D   CALL          ;TRANS. FPA1 TO FPA2
0E50          EX
0E51          DEC
0E52          LD          ;PICK UP EXPONENT
0E53          OR
0E54          RET
0E55          ADD          ;MULTIPLY BY FOUR
0E57          JP          ;IF C, OVER FLOW ERROR
0E5A          LD
0E5B          PUSH
0E5C          CALL          ;ADD DOUBLE PRECISION TO MULTIPLY BY 5
0E5F          POP
0E60          INC          ;MULTIPLY BY TWO TO MAKE MULTIPLY BY 10
0E61          RET
0E62          JP          ;OVER FLOW ERROR

;*****
;ROUTINE ENTERED @ 0E6C FROM PARSER WHEN FINDS DIGIT
;*****
0E65  ASCBIN  CALL          ;ZERO FPA1 EXP.(ROUTINE ASCII TO BINARY)
0E68          CALL          ;CHG TYPFLG TO DOUBLE
0E6B          DEFB        0F6H ;NEXT INSTRUCTION 'OR 0AFH'
0E6C          XOR        A
0E6D          EX
0E6E          LD
0E71          LD          ;ZERO OUT HL
0E72          LD
0E73          CALL          ;CALL IF FROM PARSER
0E76          EX
0E77          LD          ;GET TOKEN
0E78          CP          ;IS "-" ?
0E7A          PUSH          ;SAVE TOKEN AND FLAG
0E7B          JP

```

Math Routines: Disassembly

```

0E7E      CP                      ;IS "+" ?
0E80      JR
0E82      DEC                    ;BACK-UP TOKEN PTR
0E83  Z0E83  RST                  ;RE-GET TOKEN
0E84      JP                      ;JP IF DIGIT
0E87      CP                      ;IS IT "." ?
0E89      JP
0E8C      CP                      ;IS IT "E" ?
0E8E      JR
0E90      CP                      ;IS IT "%" ? (INTEGER)
0E92      JP
0E95      CP                      ;IS IT "#" ? (DBL)
0E97      JP                      ;CONVERT TO DBL, INC HL, RET TO 0EC7H
0E9A      CP                      ;IS IT "!" ? (SNG)
0E9C      JP                      ;CONVERT TO SNG, INC HL, RET TO 0EC7H
0E9F      CP                      ;IS IT "D" ? (DOUBLE)
0EA1      JR
0EA3      OR

;*****
;NEXT CALL CONVERTS TO SINGLE IF ENTERED FROM "E".
;OR TO DBL IF ENTERED FROM "D".
;*****
0EA4  Z0EA4  CALL
0EA7      PUSH
0EA8      LD                      ;PUT 0EBDH ON STACK FOR RET
0EAB      EX
0EAC      RST                    ;GET NEXT TOKEN
0EAD      DEC
0EAE      CP                      ;"-" FUNCTION ?
0EB0      RET
0EB1      CP                      ;"-" SIGN ?
0EB3      RET
0EB4      INC
0EB5      CP                      ;"+" FUNCTION ?
0EB7      RET
0EB8      CP                      ;"+" SIGN ?
0EBA      RET
0EBB      DEC                    ;BACKUP TOKEN PTR
0EBC      POP
0EBD  Z0EBD  RST                  ;REGET TOKEN
0EBE      JP                      ;JP IF DIGIT
0EC1      INC
0EC2      JR
0EC4      XOR
0EC5      SUB
0EC6      LD
0EC7  Z0EC7  PUSH
0EC8      LD
0EC9      SUB
0ECA  Z0ECA  CALL                ;IF POS., MULTIPLY BY 10

```

Math Routines: Disassembly

```

0ECD          CALL          ;IF NEG., DIVIDE BY 10
0ED0          JR
0ED2          POP
0ED3          POP
0ED4          PUSH
0ED5          CALL          ;FIND ABS
0ED8          POP
0ED9          RST          ;RETURN IF NOT INT.
0EDA          RET
0EDB          PUSH
0EDC          LD
0EDF          PUSH
0EE0          CALL
0EE3          RET
0EE4 Z0EE4    RST          ;DETERMINE TYPE
0EE5          INC
0EE6          JR
0EE8          CALL          ;CONVERT TO SNG
0EEB          JP
0EEE Z0EEE    RST          ;CHECK TOKEN TYPE
0EEF          JP
0EF2 Z0EF2    INC
0EF3          JR
0EF5 Z0EF5    OR
0EF6 Z0EF6    CALL
0EF9          JR
;*****
;ROUTINE CONVERTS TO SINGLE (Z SET) OR DBL (NZ SET)
;*****
0EFB Z0EFB    PUSH          ;ENTER FROM 0EA5H,0EE8H, OR 0EF6H
0EFC          PUSH          ;SAVE ALL REGISTERS
0EFD          PUSH
0EFE          PUSH          ;SAVE FLAG STATUS
0EFF          CALL          ;IF Z, CONVERT TO SNG
0F02          POP           ;RESTORE FLAG
0F03          CALL          ;IF NZ, CONVERT TO DBL
0F06          POP           ;RESTORE REGISTERS
0F07          POP
0F08          POP
0F09          RET
;*****
;MULTIPLY FPA1 BY 10
;EITHER SINGLE OR DOUBLE
;*****
0F0A Z0F0A    RET          ;RET IF Z
0F0B          PUSH          ;SAVE FLAGS
0F0C          RST          ;TEST TYPE FLAG FOR SGL OR DBL
0F0D          PUSH
0F0E          CALL          ;CALL IF SNG TO MULTIPLY BY 10
0F11          POP

```

Math Routines: Disassembly

```

0F12          CALL          ;CALL IF DBL TO MULTIPLY BY 10
0F15          POP
0F16          DEC           ;ADJUST ACCUM
0F17          RET

;*****
;DIVIDE FPA1 BY 10 EITHER SNG OR DBL
;*****
0F18 DIVTEN   PUSH
0F19          PUSH
0F1A          PUSH
0F1B          RST           ;TEST TYPE
0F1C          PUSH
0F1D          CALL          ;CALL IF SNG
0F20          POP
0F21          CALL          ;CALL IF DBL
0F24          POP
0F25          POP
0F26          POP
0F27          INC           ;ADJUST ACCUM
0F28          RET

;*****
;ROUTINE TO CONSTRUCT NUMBER FROM DIGITS PASSED 0E84H
;*****
0F29 Z0F29    PUSH
0F2A          LD            ;MAKE B NON-ZERO IF PARSER FOUND MORE
0F2B          ADC            ;THAN ONE DECIMAL POINT
0F2C          LD
0F2D          PUSH
0F2E          PUSH
0F2F          LD            ;PICK UP THE DIGIT
0F30          SUB            ;CONVERT TO BINARY
0F32          PUSH          ;SAVE FOR 0F46H
0F33          RST            ;TEST TYPE FLAG
0F34          JP            ;JP IF NOT INT
0F37          LD            ;RECOVER INT FROM FPA1
0F3A          LD            ;INIT FOR MAX NUM. (/10)
0F3D          RST            ;CP INT 3277
0F3E          JR            ;JP IF GREATER
0F40          LD            ;***
0F41          LD            ;MULTIPLY
0F42          ADD            ; VAL IN REG. HL BY 10
0F43          ADD            ;***
0F44          ADD
0F45          ADD
0F46          POP
0F47          LD            ;PUT NEW DIGIT INTO REG C
0F48          ADD            ;ADD THE VALUE
0F49          LD            ;TEST FOR OVERFLOW
0F4A          OR
0F4B          JP            ;CONVERT TO SNG IF INTEGER OVERFLOW

```

Math Routines: Disassembly

```

0F4E          LD                      ;ELSE REPLACE INT IN FPA1
0F51  Z0F51   POP
0F52          POP
0F53          POP
0F54          JP                      ;GO BACK TO RST 20H
;*****
;NUMBER OVERFLOWS INT. CONVERT TO SINGLE
;*****
0F57  Z0F57   LD                      ;PLACE DIGIT IN ACCUM
0F58          PUSH                    ; AND SAVE IT
0F59  Z0F59   CALL                    ;CVRT INT. IN FPA1 TO SNG
0F5C          SCF
0F5D  Z0F5D   JR                      ;JP IF CAME FROM 0F34H (IF NOT INT)
0F5F          LD                      ;MOVE 1E+6 TO RFPA
0F62          LD
0F65          CALL                    ;CP SINGLE PREC.
0F68          JP                      ;JP IF FPA1 > 1E+6
0F6B          CALL                    ;MULT. FPA1 BY 10
0F6E          POP
0F6F          CALL                    ;ADD DIGIT TO FPA1
0F72          JR
;*****
;NUMBER CONVERTS TO DOUBLE PRECISION IF > 1E+6
;*****
0F74  Z0F74   CALL                    ;CLEAR EXTENDED FPA1 AND SET TYP TO 8
0F77  Z0F77   CALL                    ;MULT DBL BY 10
0F7A          CALL                    ;FPA1 -> FPA2
0F7D          POP
0F7E          CALL                    ;CVRT DIGITS TO FLOATING PT IN FPA1
0F81          CALL                    ;CLEAR EXTENDED FPA1
0F84          CALL                    ;FPA2 + FPA1 -> FPA1
0F87          JR
;*****
;CONVERT DIGIT TO SINGLE AND ADD TO FPA1
;*****
0F89  Z0F89   CALL                    ;STACK FPA1
0F8C          CALL                    ;PUT DIGIT VAL AND CONVERT TO FLOATING PT
0F8F          POP                    ;RECOVER STACKED VAL
0F90          POP
0F91          JP                      ;FPA1 + RFPA -> FPA1
;*****
;ROUTINE TO CONVERT EXPONENT DIGITS TO A VALUE
;*****
0F94  Z0F94   LD                      ;PICK UP CURRENT EXPONENT
0F95          CP
0F97          JR                      ;OVERFLOW IF >= 10
0F99          RLCA                    ;***
0F9A          RLCA                    ;SINGLE BYTE MULTIPLY BY 10
0F9B          ADD                    ;***
0F9C          RLCA

```

Math Routines: Disassembly

```

0F9D      ADD
0F9E      SUB
0FA0      LD
0FA1      DEFB      OFAH      ;HIDE NEXT INSTRUCTION WITH
                                ;A JP M,321EH
0FA2      Z0FA2     LD          ;FORCE OVERFLOW ERROR (EXP TOO LARGE)
0FA4      JP
;*****
;ROUTINE TO PRINT OUT LINE NUMBER FOR TRON
;*****
0FA7      PUSH
0FA8      LD          ;PT TO " IN "
0FAB      CALL        ;OUTPUT LINE
0FAE      POP
0FAF      WRLNO     CALL        ;REG HL -> FPA1
0FB2      XOR          ;ZERO PRINT USING FLAG
0FB3      CALL        ; VIA THIS CALL
0FB6      OR
0FB7      CALL        ;CVRT FPA1 TO ASCII
0FBA      JP          ;AND PRINT OUT NUMBER
;*****
;PROCESSING OF DATA VALUES OUTPUT (FLTG PT, INT)
;INCLUDES FORMATTING
;ENTER AT 0FBDH FROM 111DH, 20C0H (PRINT), 2836H (STR$)
;ENTER AT 0FBEH FROM 2DCEH (USING)
;*****
0FBD      BINASC     XOR          ;INIT TO CLEAR USGFLG
0FBE      ASCUSG     CALL        ;INIT USGFLG WITH CONVENTS REG A
;*****
;TEXT FOR USING "+" SPECIFIER
;*****
0FC1      AND          ;"+" IS BIT 3 OF USGFLG
0FC3      JR          ;DON'T INIT '+' IF NOT SPECIFIED
0FC5      LD          ;INIT WITH '+'
0FC7      Z0FC7     EX          ;TEST IF POS OR NEG
0FC8      CALL        ;SIGN FLAG SET AS TO SIGN
0FCB      EX          ;DE NOW HAS THE FPA1 INTEGER
0FCC      JP          ;BYPASS IF POSITIVE INTEGER
0FCF      LD          ;ELSE INSERT '-'
0FD1      PUSH        ;TAKE ABS VALUE
0FD2      PUSH
0FD3      CALL
0FD6      POP
0FD7      POP
0FD8      OR
0FD9      Z0FD9     INC          ;INSERT ASCII ZERO
0FDA      LD          ;AT NEXT BUFFER POS
0FDC      LD          ;P/U USING CONTROL BYTE
0FDF      LD
0FE0      RLA          ;SET CARRY IF BIT 7 ON

```

Math Routines: Disassembly

```

0FE1          LD
0FE4          JP          ;JUMP IF USGFLG HAS BIT 7 ON
0FE7          JP          ;JUMP IF USGFLG OFF (NOT USING)
0FEA          CP          ;JUMP IF DBL OR SNGL
0FEC          JP
0FEF          LD          ;NO COMMAS OR DEC PTS, NBR IS INTEGER
OFF2          CALL        ;CONVERT INT IN FPA1 TO ASCII AT ASCBUF
OFF5  ZOFF5    LD
OFF8          LD          ;P/U 1ST BUF CHAR (SPACE)
OFF9          LD          ;SPACE
OFFB          LD
OFFE          LD          ;TEST FOR '*' FUNCTION
OFFF          AND
1001          JR          ;BYPASS IF NOT "*" SPECIFIED
1003          LD          ;XFR BUF CHAR TO A
1004          CP          ;CPR WITH SPACE
1005          LD          ; '*'
1007          JR          ;JUMP IF BUF CHAR NOT SPACE
1009          LD          ;REPL B WITH "*"
100A  Z100A    LD          ;REPL BUF CHAR WITH "*"
100B          RST         ;P/U NEXT CHAR OR
100C          JR          ;JUMP IF AT END OF BUFFER
100E          CP          ; 'E'?
1010          JR
1012          CP          ; 'D'?
1014          JR
1016          CP          ; '0'?
1018          JR
101A          CP          ; ', '?
101C          JR
101E          CP          ; '. '?
1020          JR
1022  Z1022    DEC
1023          LD          ;INSERT ASCII ZERO
1025  Z1025    LD          ;CK USGFLG FOR BIT 4
1026          AND
1028          JR          ;JP IF NOT SPECIFIED
102A          DEC         ;INSERT FLOATING $
102B          LD
102D  Z102D    LD          ;CK USGFLG FOR BIT 2
102E          AND
1030          RET         ;RETURN IF SPECIFIED
1031          DEC
1032          LD          ;INSERT '*'
1033          RET
;*****
;ROUTINE TO INIT USGFLG
;*****
1034  Z1034    LD          ;MOVE CONTENTS OF ACCUM TO USING FLAG
1037          LD

```

Math Routines: Disassembly

```

103A      LD
103C      RET
;*****
;ROUTINE TO CONVERT SNGL OR DBL TO ASCII
;*****
103D      Z103D    CP          ;SET C FLAG IF SNGL
103F      PUSH     ;SAVE BUFFER POINTER
1040      SBC      ;REDUCE TYPE LEN IF SNGL
1042      RLA      ;MUL TYPE BY 2
1043      LD        ;INC THE RESULT BY 1
1044      INC      ;RESULT IS SNGL=7, DBL=17
1045      CALL     ;CONVERT FPA1 TO PROPER RANGE
1048      LD        ;INIT B=3 FOR DEC PTS
104B      ADD      ;ADD FIELD WIDTH TO # OF PLACES SHIFTED
104C      JP       ;JUMP IF SHIFTED > FIELD WIDTH
104F      INC
1050      CP
1051      JR        ;JUMP IF NBR > FIELD
1053      INC
1054      LD B      ;SET B TO POSITION DEC PT
1055      LD
1057      Z1057    SUB
1059      POP
105A      PUSH
105B      CALL     ;INSERT ",", OR "." IF NEEDED
105E      LD        ;INSERT ASCII ZERO
1060      CALL     ;INC HL, RET
1063      CALL     ;CONVERT TO ASCII
1066      Z1066    DEC
1067      LD
1068      CP        ;'0'
106A      JR
106C      CP        ;'.','?'
106E      CALL     ;INC HL, RET
1071      POP
1072      JR        ;TEST EXPONENT FOR OUTPUT
1074      INSEXP   PUSH     ;IF ZERO, DON'T; ELSE DO 'E' OR 'D'
1075      RST      ;SAVE EXPONENT VALUE
1076      LD        ;SET C-FLAG IF SNGL, RESET IF DBL
1078      ADC      ;22H + 22H + 0 (DBL) = "D"
1079      LD        ;22H + 20H + 1 (SNGL)= "E"
107A      INC      ;INSERT "D" OR "E" INTO BUFFER
107B      POP
107C      LD        ;NEXT 3 INST INSERT EXPONENT SIGN
107E      JP       ;INSERT '+'
1081      LD        ;BYPASS IF POS
1083      CPL       ;ELSE INSERT '-'
1084      INC      ;CONVERT HEX 00-63 TO DEC 00-99
1085      Z1085    LD
1087      Z1087    INC

```

Math Routines: Disassembly

```

1088      SUB
108A      JR
108C      ADD
108E      INC
108F      LD          ;INSERT 1ST EXP DIGIT
1090      INC
1091      LD          ;INSERT 2ND EXP DIGIT
1092      Z1092      INC
1093      Z1093      LD          ;INSERT END-OF-BUFFER MARK
1095      EX
1096      LD
1099      RET

;*****
;HERE FROM 0FE4H IF USGFLG HAS BIT 7 ON
;*****
109A      Z109A      INC
109B      PUSH
129C      CP          ;CPR VARTYP TO 4
109E      LD          RECOVER USGFLG
109F      JP          ;JP IF FPA1 IS SNGL OR DBL
10A2      RRA          ;JP IF SCIENTIFIC NOTATION
10A3      JP          ;REQUESTED (USGFLG BIT 0 SET)
10A6      LD          ;INIT FOR INTEGER
10A9      CALL        ;TEST USGFLG BIT 6
10AC      POP          ;RCVR DEC PT CTR IN D
10AD      LD
10AE      SUB
10B0      CALL        ;INSERT "A" ZEROES
10B3      CALL        ;CONVERT FROM POWER-OF-TEN TABLE
10B6      Z10B6      LD          ;TEST COMMA COUNTER
10B7      OR
10B8      CALL        ;"DEC HL, RET"
10BB      DEC
10BC      CALL        ;INSERT "A" ZEROES
10BF      Z10BF      PUSH
10C0      CALL
10C3      POP
10C4      JR
10C6      LD
10C7      INC
10C8      Z10C8      LD
10CA      LD
10CD      Z10CD      INC
10CE      Z10CE      LD          ;P/U PTR TO DEC PT IN BUFFER
10D1      SUB
10D2      SUB
10D3      RET
10D4      LD
10D5      CP          ;SPACE?
10D7      JR

```

Math Routines: Disassembly

```

10D9      CP                ; '*'?
10DB      JR
10DD      DEC
10DE      PUSH
10DF      Z10DF  PUSH
10E0      LD
10E3      PUSH
10E4      RST
10E5      CP                ; '-'?
10E7      RET
10E8      CP                ; '+'?
10EA      RET
10EB      CP                ; '$'?
10ED      RET
10EE      POP
10EF      CP                ; '0'?
10F1      JR
10F3      INC
10F4      RST
10F5      JR
10F7      DEC
10F8      DEFB      1      ;HIDE NEXT 2 INST WITH 'LD BC'
10F9      Z10F9  DEC
10FA      LD
10FB      POP
10FC      JR
10FE      POP
10FF      JP
1102      Z1102  POP
1103      JR
1105      POP
1106      LD                ;INSERT '%' OVERFLOW IND
1108      RET

;*****
;HERE IF FPA1 IS SNGL OR DBL & NO SCIENTIFIC NOTATION
;*****
1109      Z1109  PUSH
110A      RRA                ;TEST USGFLG(0) FOR SCIENTIFIC
110B      JP                ;JUMP IF WANT IT
110E      JR                ;JUMP IF FPA1 WAS SNGL (TESTED @ 109C)
1110      LD                ;1D+16
1113      CALL              ;FPA2 VS 1D+16
1116      LD                ;INIT FOR 16 DIGIT FIELD
1118      JP                ;JP IF < 1D+16
111B      Z111B  POP          ;ELSE CONVERT & ADD OVRFLW
111C      POP              ;SINCE NBR EXCEEDS FIELD
111D      CALL              ;CONVERT NBR TO ASCII
1120      DEC
1121      LD                ;INSERT '%' OVRFLW IND
1123      RET

```

Math Routines: Disassembly

```

;*****
;HERE ON SNGL & NO SCIENTIFIC NOTATION
;*****
1124 Z1124 LD ;1E+16 -> BCDE
1127 LD
112A CALL ;CPR FPA1 TO 1E+16
112D JP ;JP IF < 1E+16 FOR OVRFLW
1130 LD ;INIT FOR 6 DIGIT FIELD
1132 Z1132 CALL ;TEST SIGN OF FPA1
1135 CALL ;PUT IN RANGE IF <> 0
1138 POP
1139 POP
113A JP ;JP IF SMALLER THAN RANGE
113D PUSH
113E LD
113F LD
1140 SUB
1141 SUB
1142 CALL ;"A" ZEROES INTO BUFFER
1145 CALL
1148 CALL ;CONVERT TO ASCII
114B OR
114C CALL
114F OR
1150 CALL ;CK ON ",", OR "." NEEDED
1153 POP
1154 JP

;*****
;HERE ON SNGL OR DBL, NO SCIENTIFIC, NBR < RANGE
;*****
1157 Z1157 LD
1158 LD
1159 OR
115A CALL ;DEC A, RET
115D ADD
115E JP
1161 XOR
1162 Z1162 PUSH
1163 PUSH
1164 Z1164 CALL ;IF NEG., DIV. 10
1167 JP ;IF STILL NEG., DIV 10
116A POP
116B LD
116C SUB
116D POP
116E LD
116F ADD
1170 LD
1171 JP

```

Math Routines: Disassembly

```

1174      SUB
1175      SUB
1176      CALL          ;"A" ZEROES -> BUFFER
1179      PUSH
117A      CALL
117D      JR
117F      Z117F      CALL          ;"A" ZEROES -> BUFFER
1182      LD
1183      CALL
1186      LD
1187      XOR
1188      SUB
1189      SUB
118A      CALL          ;"A" ZEROES -> BUFFER
118D      PUSH
118E      LD
118F      LD
1190      Z1190      CALL          ;CONVERT HEX TO ASCII
1193      POP
1194      OR
1195      JR
1197      LD          ;DEC PT PTR -> HL
119A      Z119A      ADD
119B      DEC
119C      CALL          ;"A" ZEROES -> BUFFER
119F      LD
11A0      JP
      ;*****
      ;HERE IF INTEGER & SCIENTIFIC NOTATION REQUESTED
      ;*****
11A3      Z11A3      PUSH
11A4      PUSH
11A5      CALL          ;CONVERT INTEGER TO SNGL
11A8      POP
11A9      XOR
      ;*****
      ;HERE IF SNGL OR DBL & SCIENTIFIC NOTATION REQUESTED
      ;*****
11AA      Z11AA      JP
11AD      LD          INIT FOR 16 DIGIT FIELD
11AF      DEFB      1      ;HIDE NEXT INST WITH 'LD BC'
11B0      Z11B0      LD          ;INIT SNGL FOR 6-DIGIT FIELD
11B2      CALL
11B5      SCF
11B6      CALL          ;CONVERT FPA1 TO RANGE IF <> 0
11B9      POP
11BA      POP
11BB      PUSH
11BC      LD
11BD      OR

```

Math Routines: Disassembly

```

11BE      PUSH
11BF      CALL          ;DEC A, RET
11C2      ADD
11C3      LD
11C4      LD          ;SET D TO -1 IF USGFLG(2)
11C5      AND          IS SET OR SET D TO +1 IF
11C7      CP          ;USGFLG(2) RESET
11C9      SBC          ;BIT 2 USED FOR '-' AT END
11CA      LD          ;OF FIELD
11CB      ADD
11CC      LD
11CD      SUB
11CE      PUSH
11CF      PUSH
11D0      Z11D0      CALL
11D3      JP
11D6      POP
11D7      POP
11D8      PUSH
11D9      PUSH
11DA      JP
11DD      XOR
11DE      Z11DE      CPL
11DF      INC
11E0      ADD
11E1      INC
11E2      ADD
11E3      LD
11E4      LD
11E6      CALL          ;CONVERT HEX TO ASCII
11E9      POP
11EA      CALL
11ED      POP
11EE      POP
11EF      CALL          ;DEC HL, RET
11F2      POP
11F3      JR
11F5      ADD
11F6      SUB
11F7      SUB
11F8      Z11F8      PUSH
11F9      CALL          ;INSERT THE EXPONENT
11FC      EX
11FD      POP
11FE      JP
;*****
;ROUTINE TO PUT FPA1 IN THE RANGE 1E+5 TO 1E+6 IF SNGL
;      FPA1 IN THE RANGE 1D+15 TO 1D+16 IF DBL
;*****
1201      Z1201      PUSH

```

Math Routines: Disassembly

```

1202      XOR
1203      PUSH
1204      RST
1205      JP          ;JUMP IF SNGL
1208  Z1208  LD
120B      CP          ;65536
120D      JP          ;JP WHEN EXP EXCEEDS 2**16
1210      LD          ;1D+10
1213      LD          ;XFR 1D+10 TO FPA2
1216      CALL
1219      CALL          ;MULTIPLY FPA1 BY 1D+10
121C      POP
121D      SUB          ;ADJUST COUNTER
121F      PUSH
1220      JR
      ;*****
      ;"      THIS PUTS FPA1 ABOVE MIN VALUE (1E+5/1D+15)
      ;*****
1222  Z1222  CALL
1225  Z1225  RST
1226      JR          ;JUMP IF DBL
1228      LD          ;1E+5 -> BCDE
122B      LD
122E      CALL          ;CPR SNGL TO 1E+5
1231      JR
1233  Z1233  LD          ;1D+15
1236      CALL          ;CPR DBL TO 1D+15
1239  Z1239  JP          ;JP IF FPA1 > MIN VALUE
123C      POP          ;ELSE RCVR COUNTER
123D      CALL          ;AND RAISE BY POWER OF TEN
1240      PUSH          ;SAVE COUNTER
1241      JR
1243  Z1243  POP          ;FPA1 EXCEEDS MAX, RCVR
1244      CALL          ;THE COUNTER AND REDUCE BY
1247      PUSH          ;A POWER OF TEN
1248      CALL
124B  Z124B  POP          ;RCVR SHIFT COUNTER
124C      OR          ;AND TEST SIGN
124D      POP          ;RCVR BUFFER POINTER
124E      RET
      ;*****
      ;THIS PUTS FPA1 BELOW MAX VALUE (1E+6/1D+16)
      ;*****
124F  Z124F  RST
1250      JP          ;JUMP IF NOT SNGL
1253      LD          ;1E+6 -> BCDE
1256      LD
1259      CALL          ;CPR SNGL TO MAX VALUE
125C      JR
125E  Z125E  LD          ;1D+16

```

Math Routines: Disassembly

```

1261          CALL          ;CPR DBL TO MAX VALUE
1264 Z1264    POP           ;POP RETURN ADDRESS
1265          JP            ;JP IF FPA1 > MAX VALUE
1268          JP            ;ELSE RETURN
          ;*****
          ;LOAD ZEROES -> ASCBUF FOR AS MANY BYTES AS COUNT IN REG A
          ;*****
1269 Z1269    OR
126A Z126A    RET
126B          DEC
126C          LD            ;INSERT ASCII ZERO
126E          INC
126F          JR
          ;*****
          ;LOADS ZEROES -> ASCBUF FOR COUNT OF REG A
          ;& INSERTS COMMAS AS NEEDED PLUS THE DECIMAL POINT
          ;*****
1271 Z1271    JR
1273 Z1273    RET
1274          CALL
1277 Z1277    LD
1279          INC
127A          DEC
127B          JR
          ;*****
          ;
          ;*****
127D Z127D    LD
127E          ADD
107F          INC
1280          LD
1281          INC
1282 Z1282    SUB
1284          JR
1286          ADD
1288          LD
1289 Z1289    LD
128C          AND            ;TEST BIT 6 FOR COMMA
128E          RET
128F          LD
1290          RET
          ;*****
          ;ROUTINE TO INSERT "," OR "." AS NEEDED
          ;*****
1291 Z1291    DEC
1292          JP

```

Math Routines: Disassembly

```

;*****
;ROUTINES TO CONVERT HEX DATA TO ASCII CHARACTERS
;*****
12A4 Z12A4  PUSH
12A5      RST
12A6      JP          ;JUMP IF SNGL
12A9      PUSH
12AA      PUSH
12AB      CALL        ;XFR DATA FROM FPA1 TO FPA2
12AE      LD          ;0.5
12B1      CALL        ;XFR FM MEM TBL TO FPA1
12B4      CALL
12B7      XOR
12B8      CALL
12BB      POP
12BC      POP
12BD      LD          ;PT TO DBL CONV TBL
12C0      LD          ;LOOP 12DE-12C2 10 TIMES
12C2 Z12C2  CALL        ;", " OR "." NEEDED?
12C5      PUSH
12C6      PUSH
12C7      PUSH
12C8      PUSH
12C9      LD          ;INIT TO COUNT THE SUBTRACTS
12CB Z12CB  INC
12CC      POP
12CD      PUSH
12CE      CALL        ;SUB MANT (HL) FM (411D)
12D1      JR
12D3      POP
12D4      CALL        ;ADD MANT (HL) TO (411D)
12D7      EX
12D8      POP
12D9      LD          ;COUNT INTO BUFFER
12DA      INC         ;PT TO NEXT ASCBUF POS
12DB      POP         ;RECOVER LOOP COUNT
12DC      POP
12DD      DEC         ;DEC LOOP COUNT
12DE      JR          ;END OF DO LOOP
12E0      PUSH
12E1      PUSH
12E2      LD
12E5      CALL        ;(HL) -> FPA1
12E8      JR
;*****
;HERE IF SINGLE PRECISION
;*****
12EA Z12EA  PUSH
12EB      PUSH

```

Math Routines: Disassembly

```

12EC          CALL          ;0.5 + FPA1
12EF          INC
12F0          CALL
12F3          CALL          ;BCDE -> FPA1
12F6  Z12F6    POP
12F7          POP
12F8          XOR
12F9          LD            ;PT TO SNGL CONV VALUES
12FC  Z12FC    CCF  C-FLG (RESET 2ND TIME LOOP)
12FD          CALL          ;CK IF '.' OR ',' NEEDED
1300          PUSH
1301          PUSH
1302          PUSH          ;SAVE BUFFER POINTER
1303          PUSH
1304          CALL          ;FPA1 -> BCDE
1307          POP
1308          LD            ;INIT ACCUM
130A  Z130A    INC          ;ACCUMULATE INTO REG B
130B          LD            ;HOW MANY TIMES THE
130C          SUB          ;CONV VALUE CAN BE
130D          LD            ;SUBTRACTED FROM THE
130E          INC          ;MANTISSA (DECIMAL)
130F          LD
1310          SBC
1311          LD
1312          INC
1313          LD
1314          SBC
1315          LD
1316          DEC          ;PT TO 1ST TABLE BYTE AGAIN
1317          DEC
1318          JR
131A          CALL          ;ADD BACK 3 BYTES (HL) + EDC
131D          INC          ;PT TO NEXT TABLE VALUE
131E          CALL          ;BCDE -> FPA1
1321          EX            ;TABLE PTR -> DE
1322          POP
1323          LD            ;INSERT ASCII VALUE INTO
1324          INC          ;BUFFER & ADVANCE POINTER
1325          POP
1326          POP
1327          JR
1329          INC          ;PT TO INTEGER TABLE @ 1000
132A          INC
132B          LD            ;ONLY 4 VALUES LEFT
132D          JR
          ;*****
          ;          CONVERT INTEGER TO ASCII
          ;*****
132F  Z132F    PUSH

```

Math Routines: Disassembly

```

1330      LD          ;POWER OF 10 TABLE
1333      LD          ;HAS 5 VALUES
1335  Z1335  CALL          ;NEED '.' OR ','?
1338      PUSH
1339      PUSH
133A      PUSH
133B      EX          ;TABLE PTR -> HL
133C      LD          ;TABLE VALUE -> STACK
133D      INC
133E      LD
133F      PUSH
1340      INC
1341      EX          ;PTR TO STACK & VALUE TO HL
1342      EX          ;TABLE VALUE -> DE
1343      LD          ;& INTEGER -> HL
1346      LD          ;INIT DECIMAL COUNTER
1348  Z1348  INC          ;ACCUM # OF TIMES WE CAN
1349  Z1349  LD          ;SUBTRACT THE CURRENT
134A      SUB          ;POWER OF TEN INTO REG B
134B      LD
134C      LD
134D      SBC
134E      LD
134F      JR
1351      ADD          ;ADD BACK ONE TIME
1352      LD          ;UPDATE INTEGER VALUE
1355      POP
1356      POP
1357      LD          ;PLACE ASCII DIGIT INTO BUF
1358      INC          ;& PT TO NEXT BUFFER POS
1359      POP          ;RECOVER DIGIT PLACE COUNTER
135A      POP
135B      DEC          ;POSITION COUNT DOWN
135C      JR          ;GO BACK FOR NEXT POSITION
135E      CALL          ;SEE IF '.' OR ',' NEEDED
1361      LD          ;INSERT HEX ZERO INTO BUFFER
1362      POP
1363      RET

```

;*****

;VARIOUS DATA VALUES & CONVERSION CONSTANTS

;*****

```

1364  Z1364  DEFW      0          ;1D+10
1366      DEFW      0
1368      DEFW      2F9H
136A      DEFW      0A215H
136C  Z136C  DEFW      0FFFDH ;1D+15
136E      DEFW      319FH
1370      DEFW      5FA9H
1372      DEFW      0B263H
1374  Z1374  DEFW      0FFFEH ;1D+16

```

Math Routines: Disassembly

```

1376      DEFW      0BF03H
1378      DEFW      1BC9H
137A      DEFW      0B60EH
137C  HALF  DEFW      0          ;0.5
137E      DEFW      0
1380  Z1380  DEFW      0
1382      DEFW      8000H
1384  Z1384  DEFW      0          ;1D+16
1386      DEFW      0BF04H
1388      DEFW      1BC9H
138A      DEFW      0B60EH
        ;*****
        ;TEN GROUPS OF 7 BYTES EACH FOR CONVERSIONS OF DBL PREC
        ;*****
138C  Z138C  DEFW      8000H      ;1D+16
138E      DEFW      0A4C6H
1390      DEFW      8D7EH
1392      DEFB      3
        ;*****
1393      DEFW      4000H      ;1D+15
1395      DEFW      107AH
1397      DEFW      5AF3H
1399      DEFB      0
        ;*****
139A      DEFW      0A000H      ;1D+14
139C      DEFW      4E72H
139E      DEFW      918H
13A0      DEFB      0
        ;*****
13A1      DEFW      1000H      ;1D+13
13A3      DEFW      0D4A5H
13A5      DEFW      0E8H
13A7      DEFB      0
        ;*****
13A8      DEFW      0E800H      ;1D+12
13AA      DEFW      4876H
13AC      DEFW      017H
13AE      DEFB      0
        ;*****
13AF      DEFW      0E400H      ;1D+11
13B1      DEFW      540BH
13B3      DEFW      2
13B5      DEFB      0
        ;*****
13B6      DEFW      0CA00H      ;1D+10
13B8      DEFW      3B9AH
13BA      DEFW      0
13BC      DEFB      0
        ;*****
13BD      DEFW      0E100H      ;1D+9

```

Math Routines: Disassembly

```

13BF      DEFW      5F5H
13C1      DEFW      0
13C3      DEFB      0
;*****
13C4      DEFW      9680H ;1D+8
13C6      DEFW      98H
13C8      DEFW      0
13CA      DEFB      0
;*****
13CB      DEFW      4240H ;1D+7
13CD      DEFW      0FH
13CF      DEFW      0
13D1      DEFB      0
;*****
;TWO CONSTANTS TO CONVERT REAL TO ASCII
;*****
13D2 Z13D2  DEFW      86A0H ;1E+6
13D4      DEFB      1
;*****
13D5      DEFW      2710H ;1E+5
13D7      DEFB      0
;*****
;POWER OF 10 TABLE FOR CONVERTING INTEGERS TO DECIMAL ASCII
;*****
13D8 Z13D8  DEFW      10000
13DA      DEFW      1000
13DC      DEFW      100
13DE      DEFW      10
13E0      DEFW      1
;*****
;ROUTINE STACKS A CALL TO CHANGE SIGN OF RESULT IN FPA1
;*****
13E2 Z13E2  LD
13E5      EX
13E6      JP
;*****
;PROCESS SQR(X) : USES POWER & EXP FUNCTIONS
;      ALGORITHM RAISES FPA1 TO 0.5 POWER
;*****
13E7 SQR     CALL          ;FPA1 -> STACK
13EA      LD              ;0.5 -> FPA1
13ED      CALL
13F0      JR
;*****
;PROCESS RAISING TO A POWER : X POWER Y
;      ALGORITHM FINDS Z SUCH THAT EXP(Z) = X POWER Y
;      I.E. Z = Y * LOG(X)
;      THEN USES EXP(X) FUNCTION TO TAKE EXP(Z)
;*****
13F2 POWER CALL          ;Y - ↑

```

Math Routines: Disassembly

```

13F5  Z13F5  POP
13F6      POP
13F7      CALL      ;POWER FOR M, Z, P
13FA      LD
13FB      JR        ;IF POWER = 0
13FD      JP
1400      OR        ;BASE FOR ZERO (POWER IS NEG)
1401  Z1401  JP        ;BY ZERO ERROR
1404  Z1404  OR
1405      JP        ;ZEROES EXPONENT & RETURNS
1408      PUSH      ;SAVE X
1409      PUSH
140A      LD
140B      OR        ;ONES TO ALL BUT SIGN BIT
140D      CALL      ;FPA1 -> BCDE
1410      JP        ;JUMP ON POS X
1413      PUSH      ;SAVE X AGAIN
1414      PUSH
1415      CALL      ;FIND INT(Y)
1418      POP        ;RECOVER X
1419      POP
141A      PUSH
141B      CALL
141E      POP
141F      LD
1420      RRA
1421  Z1421  POP      ;STACK -> FPA1
1422      LD
1425      POP
1426      LD
1429      CALL      ;INIT TO CHG SIGN OF RESULT
142C      CALL      ;CHG SIGN OF FPA1
142F      PUSH
1430      PUSH
1431      CALL      ;TAKE LOG(X)
1434      POP
1435      POP
1436      CALL      ;Y * LOG(X)

;*****
;PROCESS EXP(X) FUNCTION
;*****
1439  EXP    CALL      ;SAVE X ON STACK
143C      LD        ;1.4427 (BASE 2 LOG OF E)
143F      LD
1442      CALL      ;X * 1.4427 -> FPA1
1445      LD
1448      CP        ;128
144A      JP        ;OVRFLW IF >=127
144D      CALL      ;TAKE INT(FPA1)
1450      ADD

```

Math Routines: Disassembly

```

1452      ADD
1454      JP
1457      PUSH
1458      LD          ;1.0
145B      CALL        ;1.0 + FPA1
145E      CALL        ;.693147 * ( 1.0 + FPA1 )
1461      POP
1462      POP          ;RCVR X
1463      POP
1464      PUSH
1465      CALL          ;X - ABOVE RESULT
1468      CALL
146B      LD          ;PT TO COMP TABLE
146E      CALL        ;PERFORM SERIES CALC
1471      LD          ;RECOVER EXPONENT ONLY
1474      POP
1475      LD
1476      JP          ;& MULTIPLY BY PREV RESULT
;*****
;DATA VALUES FOR COMPUTING EXP(X)
;*****
1479  EXPTBL  DEFB      8          ;8 CONSTANTS FOR EXP POWER SERIES
147A      DEFW      2E40H        ;-1.41316E-14
147C      DEFW      7494H
147E      DEFW      4F70H        ;1.32988E-3
1480      DEFW      772EH
1482      DEFW      26EH          ;-8.30136E-3
1484      DEFW      7A88H
1486      DEFW      0A0E6H        ;0.0416574
1488      DEFW      7C2AH
148A      DEFW      0AA50H        ;-0.166665
148C      DEFW      7FAAH
148E      DEFW      0FFFFH        ;0.5
1490      DEFW      7F7FH
1492      DEFW      0            ;-1.0
1494      DEFW      8180H
1496      DEFW      0            ;1.0
1498      DEFW      8100H
;*****
;ROUTINE TO PROCESS SERIES CALCULATIONS
;      ALGORITHM COMPUTES : SIGMA C(I) * X**2I
;      EX) C1 * X**6 + C2 * X**4 + C3 * X**2
;*****
149A  Z149A  CALL          ;FPA1 -> STACK
149D      LD          ;ESTAB RET TO 'STACK * FPA1'
14A0      PUSH
14A1      PUSH          ;SAVE TABLE PTR
14A2      CALL        ;FPA1 -> BCDE
14A5      CALL
14A8      POP

```

Math Routines: Disassembly

```

14A9 Z14A9 CALL ;FPA1 -> STACK
14AC LD ;P/U CTR TO TABLE ENTRIES
14AD INC ;PT TO NEXT TABLE VALUE
14AE CALL ;VALUE TO FPA1
14B1 DEFB 6 ;HIDE NEXT INST WITH 'LD B'
14B2 Z14B2 POP ;RECOVER TABLE COUNTER
14B3 POP ;STACK TO BCDE
14B4 POP
14B5 DEC ;DEC CTR & RET IF LAST VALUE
14B6 RET
14B7 PUSH ;BCDE -> STACK
14B8 PUSH
14B9 PUSH ;SAVE TABLE COUNTER
14BA PUSH ;SAVE PTR TO NEXT VALUE
14BB CALL ;BCDE * FPA1 -> FPA1
14BE POP ;NEXT TABLE VALUE -> BCDE
14BF CALL
14C2 PUSH ;& SAVE PTR TO NEXT VALUE
14C3 CALL
14C6 POP ;RESTORE TABLE PTR & LOOP
14C7 JR

;*****
; PROCESS RND(X) FUNCTION
;*****
14C9 RND CALL ;CVRT ARG TO INTEGER
14CC LD ;PROVIDE ERROR IF PARM
14CD OR ;EXCEEDS INTEGER BOUNDS
14CE JP ;OR IS NEG (FC ERROR)
14D1 OR ;IF PARM IS ZERO, BYPASS NEXT
14D2 JP ;PIECE TO RETURN (0-1)

;*****
;PARM IS > 1; RETURN RND NBR (1-PARM)
;*****
14D5 PUSH
14D6 CALL ;GEN RND NBR (0-1)
14D9 CALL ;FPA1 -> RFPA
14DC EX ;PUT THE RNDNBR (0-1) ONTO
14DD EX ;THE STACK & RECOVER THE
14DE PUSH ;INTEGER IN REG HL
14DF CALL ;CVRT HL TO SNGL IN FPA1
14E2 POP ;RCVR (0-1) -> RFPA
14E3 POP
14E4 CALL ;MULT (0-1) BY PARM
14E7 LD
14EA CALL ;ADD ONE TO ABOVE RESULT -> FPA1
14ED JP ;FIND INTEGER PART OF FPA1
; & CVRT TO SINGLE PREC

;*****
; GENERATE A RANDOM NUMBER BETWEEN 0 AND 1
;*****

```

Math Routines: Disassembly

```

14F0 RND0    LD      ;PT TO MULTIPLIER CONSTANT
14F3        PUSH    ;AND SAVE PTR FOR NOW
14F4        LD      ;ZERO OUT MANTISSA RFPA
14F7        LD
14F8        LD      ;INIT TO MULT 3 BYTES
14FA Z14FA   LD      ;INIT FOR 8 BITS PER BYTE
14FC Z14FC   EX      ;*****
14FD        ADD     ;MULT VALUE IN DE BY 2
14FE        EX      ;*****
14FF        LD      ;*****
1500        RLA     ;MULT VALUE IN REG C BY 2
1501        LD      ;*****
1502        EX      ;EXCHANGE "SEED"
1503        LD      ;MULTIPLYING BY CONTENTS OF
1504        RLCA     ;4090, OR 4091, OR 4092
1505        LD      ;BY 2
1506        EX      ;EXCH BACK TO P/U COUNTER
1507        JP      ;JUMP IF END OF THIS ITERATION
150A        PUSH    ;SAVE LOOP COUNTER
150B        LD      ;*****
150E        ADD     ;ADD THE 3-BYTE SEED AT
150F        EX      ;40AAH - 40ACH TO THE
1510        LD      ;RFPA MANTISSA
1513        ADC
1514        LD      ;*****
1515        POP      ;POP LOOP COUNTER
1516 Z1516   DEC     ;DECREMENT BIT LOOP
1517        JP      ;GO BACK IF < 8
151A        EX      ;*****
151B        INC     ;ADVANCE 4090 -> 4091 -> 4092
151C        EX      ;*****
151D        DEC     ;DECREMENT BYTE COUNTER
151E        JP      ;GO BACK IF < 3
1521        POP     ;POP TO MAINTAIN STACK INTEGRITY
1522        LD      ;*****
1525        ADD     ;ADD 372837 TO MANTISSA
1526        LD      ;IN 2 STEP OPERATION.
1529        CALL    ;CHG TYPFLG TO SNGL
152C        LD      ;REST OF 3 BYTE ADD
152E        ADC
152F        LD
1532        EX
1533        LD      ;1/2 -> EXPONENT
1535        LD
1538        LD
1539        DEC
153A        LD
153B        LD
153C        LD
153E        JP

```

Math Routines: Disassembly

```

;*****
;PROCESS COS(X) : USES SIN(X)
;*****
1541 COS      LD          ;1.5708 (PI/2)
1544          CALL        ;(PI/2) + FPA1 -> FPA1
;*****
;PROCESS SIN(X) FUNCTION
;*****
1547 SIN      CALL        ;FPA1 -> STACK
154A          LD          ;6.28319 (2 PI)
154D          LD
1550          CALL        ;BCDE -> FPA1
1553          POP
1554          POP          ;RECOVER X
1555          CALL        ;X / 2PI
1558          CALL        ;X / 2PI -> STACK
155B          CALL        ;TAKE INT(X/2PI)
155E          POP
155F          POP          ;RECOVER X/2PI
1560          CALL        ;(X/2PI)-INT(X/2PI) -> FPA1
1563          LD          ;0.25
1566          CALL        ;0.25 - ABOVE RESULT
1569          CALL        ;TEST FOR M, Z, P
156C          SCF
156D          JP          ;BYPASS IF POS
1570          CALL        ;ELSE 0.5 + RESULT
1573          CALL        ;TEST FOR M, Z, P
1576          OR
1577 Z1577    PUSH
1578          CALL        ;CHG SIGN OF FPA1
157B          LD          ;0.25
157E          CALL        ;0.5 + FPA1 -> FPA1
1581          POP
1582          CALL
1585          LD          ;PT TO SERIES TABLE
1588          JP          ;& CALC SERIES
;*****
;DATA VALUES TO COMPUTE SIN(X)
;*****
158B HALFPI   DEFW        0FDBH ;1.5708 (PI/2)
158D          DEFW        8149H
158F QUARTR   DEFW        0      ;0.25
1591          DEFW        7F00H
1593 SINTBL   DEFB        5      ;5 CONSTANTS FOR SIN TABLE
1594          DEFW        0D7BAH ;39.7107
1596          DEFW        861EH
1598          DEFW        2664H ; -76.575
159A          DEFW        8799H
159C          DEFW        3458H ;81.6022
159E          DEFW        8723H

```

Math Routines: Disassembly

```

15A0      DEFW      5DE0H      ; -41.6022
15A2      DEFW      86A5H
15A4      DEFW      0FDAH      ; 6.28319 (2 PI)
15A6      DEFW      8349H

;*****
;PROCESS TAN(X) : AS SIN(X)/COS(X)
;*****
15A8      TAN       CALL              ; FPA1 -> STACK
15AB      CALL              ; CALC SIN(X)
15AE      POP                ; RECOVER X
15AF      POP
15B0      CALL              ; SIN(X) -> STACK
15B3      EX
15B4      CALL              ; X -> FPA1
15B7      CALL              ; CALC COS(X)
15BA      JP         ; SIN(X)/COS(X)

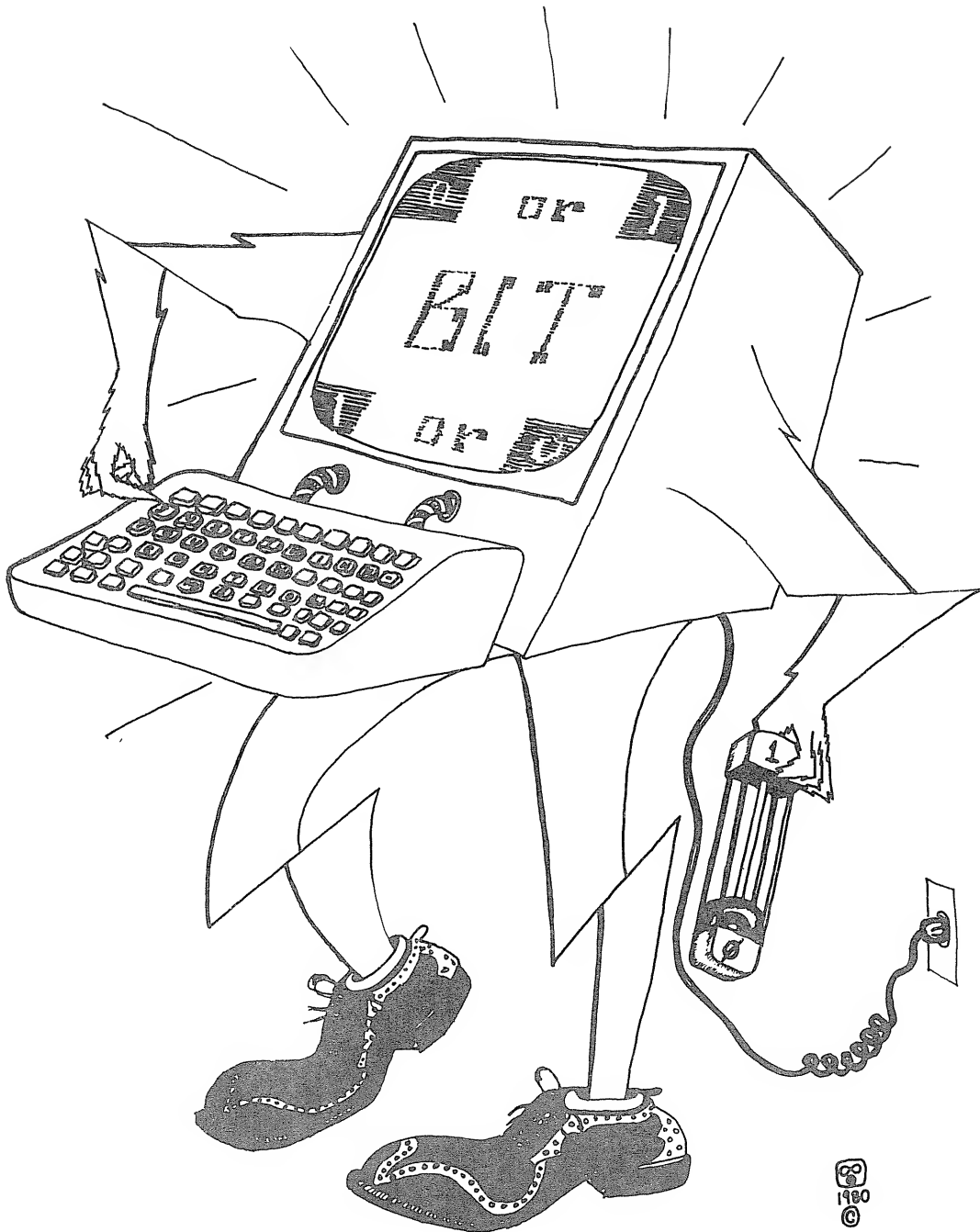
;*****
;PROCESS ATN(X) FUNCTION
;*****
15BD      ATN       CALL              ; TEST X FOR M, Z, P
15C0      CALL              ; INIT TO CHG SIGN OF RESULT
15C3      CALL              ; X < 0, SO MAKE POS NOW
15C6      LD         ; GET EXPONENT
15C9      CP         ; 1.0
15CB      JR         ; JUMP IF < ZERO
15CD      LD         ; 1.0 -> BCDE
15D0      LD
15D1      LD
15D2      CALL              ; 1.0 / X
15D5      LD         ; ESTAB RET ( PI/2 - FPA1 )
15D8      PUSH
15D9      Z15D9      LD         ; PT TO TABLE
15DC      CALL              ; CALC SERIES : C(1) + X**21
15DF      LD
15E2      RET           ; TO 710H (PI/2 - FPA1)

;*****
;DATA VALUES TO COMPUTE ATN(X)
;*****
15E3      ATNTBL    DEFB      9      ; 9 CONSTANTS FOR ATN POWER SERIES
15E4      DEFW      0D74AH      ; 2.86623E-3
15E6      DEFW      783BH
15E8      DEFW      6E02H      ; -0.0161657
15EA      DEFW      7B84H
15EC      DEFW      0C1FEH      ; 0.0429096
15EE      DEFW      7C2FH
15F0      DEFW      3174H      ; -0.0752896
15F2      DEFW      7D9AH
15F4      DEFW      3D84H      ; 0.106563
15F6      DEFW      7D5AH
15F8      DEFW      7FC8H      ; -0.142089

```

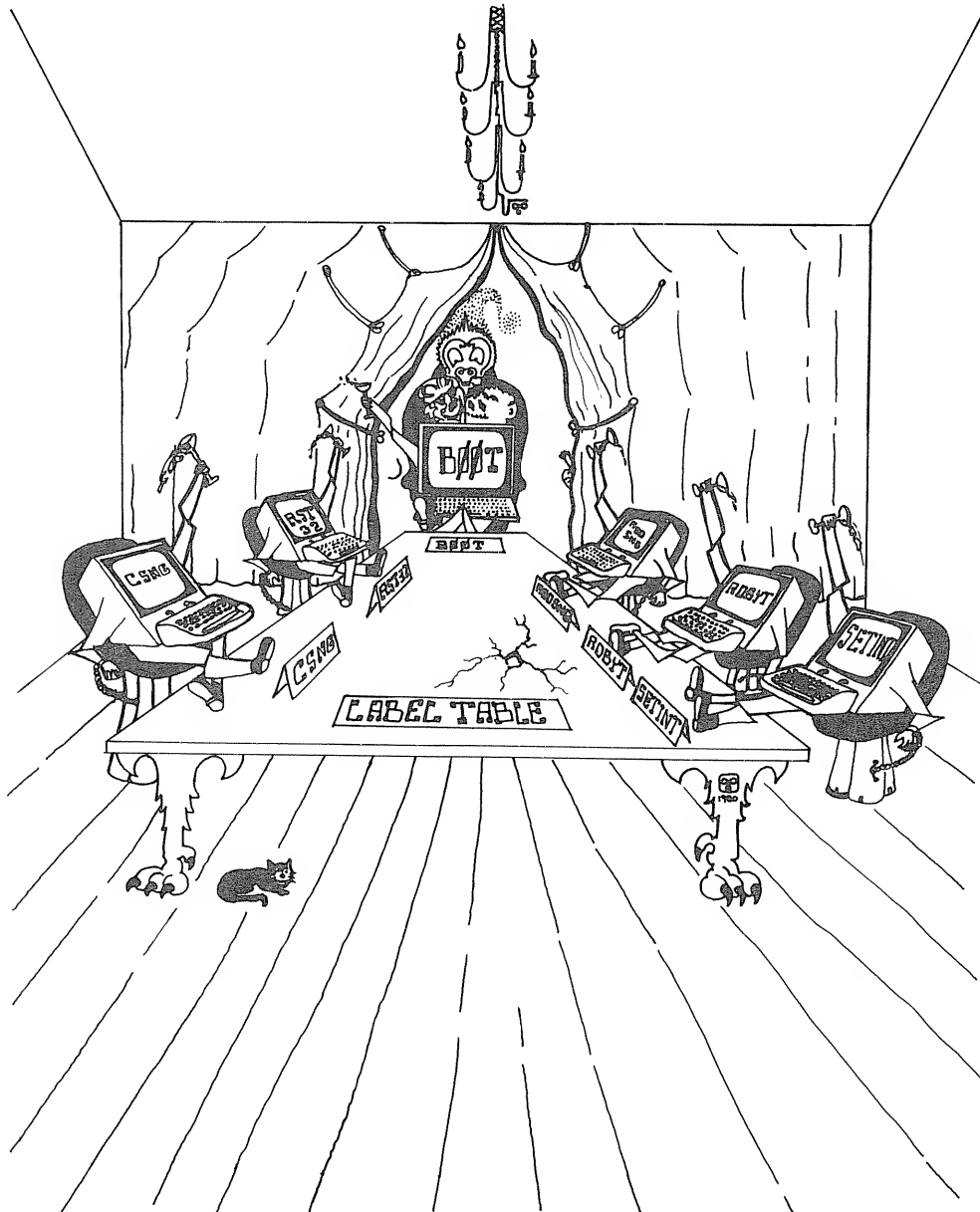
Math Routines: Disassembly

15FA	DEFW	7E91H	
15FC	DEFW	0BBE4H	;0.199936
15FE	DEFW	7E4CH	
1600	DEFW	0AA6CH	;:0.333331
1602	DEFW	7FAAH	
1604	DEFW	0	;1.0
1606	DEFW	81H	



APPENDIX A: LABEL TABLE

The following list was developed to supply the assembly language programmer with a quick reference to routine entry points, I/O areas, storage areas, and pointers. It was not designed as a complete interfacing guide. Labels listed for the various addresses provide a meaningful code-word giving some indication of the use(s) of the routines or areas. Address locations not described in this volume are either self-explanatory, may be found in Radio Shack reference manuals, or are discussed in other volumes.



<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
0000	2FFF	L2ROM	Radio Shack Level II BASIC ROM
0000		CB00T	ROM Level II Bootstrap
0008		RST8	(Parser) CP (Syntax)/RST16 if /=Else SNERR
000B		WHERE	Resolve Relocation Address
000D		DB00T	Vector to disk bootstrap
0010		RST16	INC HL. If (HL) is ASCII 0-9 SCF. If value is zero, set Z flag. Skips spaces.
0013		INBYT	Input a byte from a device
0018		RST24	CP HL,DE (A lost.)
001B		OUTBYT	Output a byte to a device
0020		RST32	P/U TYPFLG at 40AFH. If <8 SCF. RRT. Flags set as a result of type. M=Int.,Z=Str,PO=SNG,NC=DBL
0023		CTLBYT	Output a control byte to a device.
0028		RST40	JP DOS command processor
002B		KBSCAN	Keyboard scan return input in A. (DE lost.)
0030		RST48	Debug breakpoint
0033		CRTBYT	Display byte in 'A' at cursor (DE lost)
0038		RST56	Interrupt Mode 1
003B		LPTBYT	Send byte in 'A' to printer (DE lost)
0040		BUFFNV	Vector to buffer input routine (BUFFIN)
0046		DRIVRV	Vector to I/O driver routine @ 03C2H
0049	004F	GETCHR	Scan keyboard waiting for input. (DE lost)
0050	005F	KBTBL	Table of Special Characters for keyboard routine
0060	0065	DELAY	Delay routine (BC=Counter. 14.66 msec/loop)
0066	0074	NMI	Non-maskable interrupt
0075		CSTLII	Cold start for Level II BASIC
00C4	00D5	MEMSIZ	Determine memory size
0105	0110	DMEMSZ	Data "MEMORY SIZE"
0111	012B	DRSL2B	Data "RADIO SHACK LEVEL II BASIC<CR>"
012D		L3ERR	Level III error
0132	01C8	GRPHCS	Graphics Routines
0132		POINT	Point (Bcmd C6H)
0135		SET	Set (Bcmd 83H)
0138		RESET	Reset (Bcmd 82H)
019D		INKEY	Inkey\$ (Bcmd C9H)
01C9		CLS	CLS (Bcmd 84H)
01D3		RANDOM	Random (Bcmd 86H)
01D9	01F7	CWBIT	Write bit to cassette
01F8		CTOFF	Cassette off
01FE		CTON	Cassette on
0212	021D	DEFDRV	Define cassette drive from 'A'
021E	022B	CLRCFF	Clear CFF
0221		STATFF	Change status of CFF from HL
022C	0234	CSTAR	Change star in corner for cassette operations
0235	0240	CRBYTE	Read byte from cassette
0241	0260	CRBIT	Read bit from cassette
0261	0283	CW2BYT	Write byte to cassette twice
0264		CWBYT	Write byte to cassette
0284	0292	CTONWL	Cassette on, write leader and sync. byte
0287		CWLDR	Write leader and sync. byte
0293	02A8	CTONRL	Cassette on, find sync., put stars in corner

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
0296		CRLDR	Find sync., put stars in corner
029F		CSTARS	Put stars in corner
02A9		GSYSTR	Get transfer address for system
02B2		SYSTEM	System entry point
0314	031C	GETADR	Get a 2 byte address from tape (Ret in HL)
031D	0329	SYSGO	Jump to system start address
032A	0347	DSPCHR	Display byte on current device (Device flg @ 409CH)
033A		CRTOUT	Output 'A' to video (DE saved)
0348	0357	POSIND	Line position indicator
0358	0360	KBDSCN	Scan keyboard. (DE NOT LOST)
0361	0383	INCHRS	Input up to 240 chars. into 'HL' buffer. End of line has zero byte.
0384	038A	GTDCHR	Get one char. input from keyboard. (DE saved)
038B	039B	RSTDEV	Reset devices. Set output back to CRT
039C	03C1	LPDCHR	Output byte in 'A' to printer (DE saved)
03C2	03E2	DRIVER	I/O Driver
03E3	0457	KEYIN	Keyboard scan driver
0458	058C	VIDEO	Video display driver
058D	05D8	LPTDRV	Printer driver
05D1		PSTATU	Test printer status. Z Flag set if ready.
05D9	0673	BUFFIN	Buffer input routine
0674	06CF	COLDST	Cold Start
069F		DISKBT	Disk bootstrap
06CC		BASIC	Proper re-entry to Level II BASIC
06D2	06DD	RSTRTS	RST's loaded into RAM starting @ 4000H
0708		ADHALF	FPA1 + 0.5 -> FPA1
070B		ADDHL	(HL) + FPA1 -> FPA1
0710		SUBHL	(HL) - FPA1 -> FPA1
0713		SUBSNG	Subtract single precision
0716		ADDSNG	Add single precision
0778		FPA1EZ	Zero exponent of FPA1
07B2		OVERR	Overflow error
07F8	07FB	ONE1	SNG: 1.0
07FD	0800		SNG: .598979
0801	0804		SNG: .981471
0805	0808		SNG: 2.88539
0809		LOG	Log (Bcmd DFH)
0814	0819	SQR202	SNG: .707107 (SQR(2)/2) into BCDE
0834	0839		SNG: -.5 into BCDE
0841	0846		SNG: .693147 into BCDE
0847		MULSNG	Multiply single precision
0897		DIV10	FPA1 / 10 -> FPA1
08A0		POPFP	Restores old BCDE from stack
08A2		DIVSNG	Divide single precision
0955	0963	CKRMZP	Tests values for Minus, Zero, or Plus
0977		ABS	ABS (Bcmd D9H)
0982		CHGSGN	Change sign routine
098A		SGN	SGN (Bcmd D7H)
098D		SGNAE	Alternate entry point to SGN
09A4	09B0	STKFP1	Puts a real value onto the stack
09B1		HLFPA1	(HL) --> FPA1

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
09B4		SNGFPA	BCDE (Single precision val.) --> FPA1
09BF		LDFPA1	Load FPA1 into BCDE
09C2		LDFPHL	Load real value pointed to by HL
09CB		FPAMEM	Transfer FPA1 to (HL)
09D2		MOVTDI	Move data from (HL) --> (DE)
09D3		MOVTHL	Move data from (DE) --> (HL)
0A0C		CPRSNG	Compare single precision
0A39		CPRINT	Integer compare
0A78		CPRDBL	Double precision compare
0A7F		CINT	CINT (Bcmd EFH)
0A7F		USRINP	Put 'USR' function argument in HL
0A9A		SAVINT	Save integer in HL to FPA1. Vartyp -> Int (2)
0A9A		USROUT	Make HL output of 'USR' call
0A9D		SETINT	Change TYPFLG to INT
0AA3	0AA8	MINVAL	SNG: -32768 / BCDE
0AB1		CSNG	CSNG (Bcmd F0H)
0AB9		DBLSNG	Convert double to single
0ACC		SNGINT	Convert integer to single
0ACF		HL SNG	Convert HL to single
0ADB		CDBL	CDBL (Bcmd F1H)
0AEC		SETDBL	Change type flag to DBL
0AEF		SETSNG	Change type flag to single
0AF4		CHKSTR	Check type for string and TMERR if not
0AF6		TMERR	Type mismatch error
0B26		FIX	Fix (Bcmd F2H)
0B37		INT	Int (Bcmd D8H)
0B3D	0B58	INTSNG	Take integer of single
0B59	0B9D	INTDBL	Take integer of double
0BC7		SUBINT	Integer subtract
0BD2		ADDINT	Integer add
0BF2		MULINT	Integer multiply
0C5B	0C6F	ABSINT	Take absolute value of integer
0C70		SUBDBL	Subtract double
0C77		ADDDBL	Add double
0D33	0D44	DBLMA	Double precision mantissa addition
0D45	0D56	DBLMS	Double precision mantissa subtract
0DA1		MULDBL	Double precision multiply
0DD4	0DD8	TENDBL	DBL: 10.0
0DE5		DIVDBL	Double precision division
0E65		ASCBIN	Convert ASCII buffer to binary value
0E6C		ASCINT	Convert ASCII buffer to integer value
0F18		DIVTEN	Divide by ten (10)
0FAF		WRLNO	Write current line number to video
0FBD		BINASC	Convert binary value to ASCII
0FBE		ASCUSG	Convert ASCII from 'USING' routine
1364	136B		DBL: 1D+10
136C	1373		DBL: 1D+15
1374	137B		DBL: 1D+16
137C	1384	HLFDBL	DBL: .5
1384	138B		DBL: 1D+16
13D8	13E1	P10TAB	Power of ten table: 10000,1000,100,10,1
13E7		SQR	SQR (Bcmd DDH)

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
13F2		POWER	Raise to a power (Ex: X raised to the N, X**N)
1439		EXP	Exp (Bcmd E0H)
143C	1441		SNG: 1.4427
1479	1499	EXPTBL	Exp data table
147A	147D		SNG: -1.41316E-4
147E	1481		SNG: 1.32988E-3
1482	1485		SNG: -8.30136E-3
1486	1489		SNG: .0416574
148A	148D		SNG: -0.166665
148E	1491	HALF	SNG: .5
1492	1495	NEGONE	SNG: -1.0
1496	1499	ONE2	SNG: 1.0
14C9		RND	Rnd (Bcmd DEH)
1541		COS	Cos (Bcmd E1H)
1547		SIN	Sin (Bcmd E2H)
154A	154F	TWOPI	SNG: 6.28319 (2 PI) / BCDE
158B	15A7	SCDTBL	Sin/Cos data table
158B	158E	HALFPI	SNG: 1.5708 (PI/2)
158F	1592	QUARTR	SNG: .25
1593	15A7	SINTBL	Sin data table
1594	1597		SNG: 39.7107
1598	159B		SNG: -76.575
159C	159F		SNG: 81.6022
15A0	15A3		SNG: -41.3417
15A4	15A7	TWOPI	SNG: 6.28319 (2 PI)
15A8		TAN	Tan (Bcmd E3H)
15BD		ATN	Atn (Bcmd E4H)
15E3	1607	ATNTBL	Arctan data table
15E4	15E7		SNG: 2.86623E-03
15E8	15EB		SNG: -0.0161657
15EC	15EF		SNG: 0.0429096
15F0	15F3		SNG: -0.0752896
15F4	15F7		SNG: 0.106563
15F8	15FB		SNG: -0.142089
15FC	15FF		SNG: 0.199936
1600	1603		SNG: -0.333331
1604	1607	ONE3	SNG: 1.0
1608		FUNTBL	Function Table
1650	1820	BCTBL	BASIC command table (b7 of 1st char. of reserved word high)
1822		CMDTBL	Entry points for command table (BCTBL)
189A		HRCHY	Algebraic heirarchy table
18C9	18F6	ERRTBL	Error abbreviation table
1928	192E	DREADY	Data "READY<CR>"
1930	1934	DBREAK	Data "Break"
1963		CHKMEM	Check if enough memory available
197A		OMERR	Out of memory error
198A		NRERR	No resume error
1997		SNERR	Syntax error
199A		DOERR	Division by zero error
199D		NFERR	Next without For error
19A0		RWERR	Resume without error

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
19A2		ERRPRT	Output an error msg
1A19		ENTLII	Entry point to Level II BASIC
1A25		READY	Load "READY" message
1A5A		AT00FF	Turn AUTO off
1A60		AUTOON	INC to new AUTO line number
1A76		NOAUTO	Auto-off line input
1B49		NEW	New (Bcmd BBH)
1B4D		INIT	Initialize work area
1BB3		QINPUT	Print "? ". Input up to 240 characters
1BC0		SPACK	Source pack routine
1C90	1C95	RST24	CP HL,DE (A lost)
1C96	1CA0	RST8	(Parser) CP (Syntax). RST16 if equal. Else SNERR.
1CA1		FOR	For (Bcmd 81H)
1D78	1D90	RST16	Inc HL/ If (HL) is ASCII 0-9 SCF. If byte at HL=zero set Z flg. Routine skips spaces.
1D91		RESTOR	Restore (Bcmd 90H)
1DA9		STOP	Stop (Bcmd 94H)
1DAE		END	End (Bcmd 80H)
1DE4		CONT	Cont (Bcmd B3H)
1DE9		CNERR	Can't continue error
1DF7		TRON	Tron (Bcmd 96H)
1DF8		TROFF	Troff (Bcmd 97H)
1E00		DEFSTR	Defstr (Bcmd 98H)
1E03		DEFINT	Defint (Bcmd 99H)
1E06		DEFSNG	Defsnng (Bcmd 9AH)
1E09		DEFDBL	Defdbl (Bcmd 9BH)
1E3D		CKA2Z	Check if a character A-Z
1E4A		FCERR	Illegal function call error
1E4F	1E79	GETLN	Scan line for line number
1E4F	1E79	GTLNUM	Get line number
1E5A		CONVRT	Convert bytes in buffer to two-byte DE value
1E7A		CLEAR	Clear (Bcmd B8H)
1EA3		RUN	Run (Bcmd 8EH)
1EB1		GOSUB	Gosub (Bcmd 91H)
1EC2		GOTO	Goto (Bcmd 8DH)
1ED9		ULERR	Undefined line error
1EDE		RETURN	Return (Bcmd 92H)
1EEA		RGERR	Return without Gosub error
1F05		DATA	Data (Bcmd 88H)
1F07		ELSE	Else (Bcmd 95H)
		REM	Rem (Bcmd 93H)
1F21		LET	Let (Bcmd 8CH)
1F6C		ON	On (Bcmd A1H)
1FAF		RESUME	Resume (Bcmd 9FH)
1FF4		ERROR	Error (Bcmd 9EH)
2003		UEERR	Unprintable error
2008		AUTO	Auto (Bcmd B7H)
2039		IF	If (Bcmd 8FH)
2067		LPRINT	Lprint (Bcmd AFH)
206F		PRINT	Print (Bcmd B2H)
20FE		OUTCR	Output a CR to current device

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
2137		TAB	Tab((Bcmd BCH)
2169	2177	COFF10	If cassette is on, turn off
2178	217D	DRED0	Data "?RED0"
218A		FDERR	Bad file data error
219A		INPUT	Input (Bcmd 89H)
21EF		READ	Read (Bcmd 8BH)
2212		ODERR	Out of data error
227C	2285	PEXTIG	Load "?Extra ignored"
2286	2294	DEXTIG	Data "?Extra ignored"
22A0		ODERR2	Out of data error (also @ 2212H)
22BC		NEXT	Next (Bcmd 87H)
2490		DIVINT	Integer divide
249F		ADD	+ (Bcmd CDH)
249F		FNSCAN	Scan for functions
24A0		MOERR	Missing operand error
24CF		ERR	Err (Bcmd C3H)
24DD		ERL	ErI (Bcmd C2H)
24ED		VARPTR	Varptr (Bcmd C0H)
2532		SUB	- (Bcmd CEH)
25D9		RST32	From RST 32: P/U flag @ 40AFH. If <8 SCF, RRT.
25F7		OR	Or (Bcmd D3H)
25FD		AND	And (Bcmd D2H)
2608		DIM	Dim (Bcmd 8AH)
2733		DDERR	Redimensioned array error
273D		BSERR	Subscript out of range error
27C9		MEM	Mem (Bcmd C8H)
27D4		FRE	Fre (Bcmd DAH)
27F5		POS	Pos (Bcmd DCH)
27FE		USR	Usr (Bcmd C1H)
2831		IDERR	Illegal direct error
2836		STR	Str\$ (Bcmd F4H)
28A1		STERR	String formula too complex error
28A7		OUTLN	Output a line until zero (0)
28DB		OSERR	Out of string space error
298F		ADDSTR	Concatenate two strings
29A3		LSERR	String too long error
2A03		LEN	Len (Bcmd F3H)
2A0F		ASC	Asc (Bcmd F6H)
2A1F		CHR	Chr\$ (Bcmd F7H)
2A2F		STRING	String\$ (Bcmd C4H)
2A61		LEFT	Left\$ (Bcmd F8H)
2A91		RIGHT	Right\$ (Bcmd F9H)
2A9A		MID	Mid\$ (Bcmd FAH)
2AC5		VAL	Val (Bcmd F5H)
2AEF		INP	Inp (Bcmd DBH)
2AFB		OUT	Out (Bcmd A0H)
2B01		STEP	Step (Bcmd CCH)
2B29		LLIST	LList (Bcmd B5H)
2B2E		LIST	List (Bcmd B4H)
2B75		MSGOUT	Output a msg until zero (0)
2B7E		STFUNP	Scan text until zero. Unpack into INBUFP buffer

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
2BC6		DELETE	Delete (Bcmd B6H)
2BF5		CSAVE	CSave (Bcmd BAH)
2C1F		CLOAD	Cload (Bcmd B9H)
2C8A	2C92	PBAD	Prints "BAD" on screen
2CA5	2CA8	DBAD	Data "BAD<CR>"
2CAA		PEEK	Peek (Bcmd E5H)
2CB1		POKE	Poke (Bcmd B1H)
2CBD		USING	Using (Bcmd BFH)
2E60		EDIT	Edit (Bcmd 9DH)
2FC4		NOT	Not (Bcmd CBH)
37DE		COMSTA	Communication Status Address
37DF		COMDAT	Communication Data Address
37E0		INTLAT	Interrupt Latch Address
37E1		DSELECT	Disk drive select latch address
37E4		CSELECT	Cassette select latch address
37E8		LPTADR	Line printer address
37EC		FDCADR	Floppy disk controller address
37ED		TRKREG	Floppy disk track register
37EE		SECREG	Floppy disk sector register
37EF		DATREG	Floppy disk data register
37F0	37FF		Same as 37E0-37EF
3800	3BFF	KEYMEM	Keyboard memory (1,2,4,8,10,20,40,80H)
3801		KB1	Location for: @ A B C D E F G
3802		KB2	Location for: H I J K L M N O
3804		KB3	Location for: P Q R S T U V W
3808		KB4	Location for: X Y Z
3810		KB5	Location for: 0 1 2 3 4 5 6 7
3820		KB6	Location for: 8 9 : ; , - . / (Also ()*+<=>?)
3840		KB7	Location for: Enter Clear Break Arrow D.Arrow L.Arrow R.Arrow Space
3880		SHIFT	Location for: Shift (Electric pencil control key @ 10H)
3C00	3FFF	CRTMEM	Video display memory
3C00	3C3F	CRTR1	Row 1 on CRT
3C40	3C7F	CRTR2	Row 2
3C80	3CBF	CRTR3	Row 3
3CC0	3CFF	CRTR4	Row 4
3D00	3D3F	CRTR5	Row 5
3D40	3D7F	CRTR6	Row 6
3D80	3DBF	CRTR7	Row 7
3DC0	3DFF	CRTR8	Row 8
3E00	3E3F	CRTR9	Row 9
3E40	3E7F	CRTR10	Row 10
3E80	3EBF	CRTR11	Row 11
3EC0	3EFF	CRTR12	Row 12
3F00	3F3F	CRTR13	Row 13
3F40	3F7F	CRTR14	Row 14
3F80	3FBF	CRTR15	Row 15
3FC0	3FFF	CRTR16	Row 16
4000	4014	L2VECS	Level II fixed RAM vectors
4000		RST8	RST8: 1C96; (Parser) CP (Syntax)/RST16 IF=/Else SNERR
4003		RST16	RST16: 1D78; INC HL/If ASCII 0-9 SCF/Set if Z/Skip Spa

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
4006		RST24	RST24: 1C90; CP HL,DE (A lost.)
4009		RST32	RST32: 25D9; If TYPFLG<8, SCF/RRT/M=INT,Z=STR,PO=SNG,NC=DBL
400C		RST40	RST40: DOS Command Processor
400F		RST48	RST48: Debug breakpoint
4012		RST56	RST56: Interrupt mode 1
4015	401C	KEYDCB	Keyboard DCB
4015		KBTYP	DCB Type (01)
4016	4017	KBDADR	Driver address (03E3H)
4018	401C	KBCONS	Constant: 0 0 0 K I
401D	4024	CRTDCB	Video DCB
401D		CRTTYP	DCB Type (07)
401E	401F	CRTADR	Driver address (0458H)
4020	4021	CURPOS	Cursor position on screen (L,H)
4022		CURCHR	Cursor character
4023	4024	CRTCON	Constant: D 0
4025	402C	LPTDCB	Lineprinter DCB
4025		LPTTYP	DCB Type (06)
4026	4027	LPTADR	Driver address (058DH)
4028		LPTLPP	Number of lines/page
4029		LPTLCT	Line counter
402A	402C	LPTCON	Constant: 0 P R
402D	402F	DOSVEC	DOS Transfer Vector
4030	4032	ABORT	ABORT under DOS (unused under LII)
4033	4035	IODERR	Called by driver after illogical driver call
4036	403C	KBIMAG	Keyboard image
4036		KBIM1	01H
4037		KBIM2	02H
4038		KBIM3	04H
4039		KBIM4	08H
403A		KBIM5	10H
403B		KBIM6	20H
403C		KBIM7	40H
403D		CSTATU	Cassette status byte
403E	403F		Unused under Level II
4040		RTSC	25 MSec Real-time scheduling counter
4041		SECS	Seconds
4042		MINS	Minutes
4043		HRS	Hours
4044		YR	Year
4045		DAY	Day
4046		MO	Month
4047	4048	LOW	Contains address of lowest byte of avail. mem under DOS
4049	404A	DOSMEM	DOS memory size determined at power-up
404B		INTMSK	Interrupt mask
404C	404F		Unused under Level II (interrupt processing under DOS)
404C		INTENB	Interrupts enabled (bit mask)
404D	405C	INTTBL	Interrupt jump address for interrupts 0-7
4052	4053	COMINT	Communications interrupt vector
405D		DEBUG1	Debug: A or H (ASCII or H) or LSB of first breakpoint
405E		DEBUG2	Debug: 0=Normal screen, <>0 = Full screen: or MSB of first BREAKPT

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
405F		DEBUG3	Debug: Instruction byte at breakpoint
4060	4061	DEBUG4	Debug: Second breakpoint or single-step
4062		DEBUG5	Debug: Instruction byte at second breakpoint
4063	4064	DEBUG6	Debug: Address currently being displayed on screen
4065	407C	DEBUG8	DEBUG: Register save area (AF,BC,DE,HL,AF',BC',DE',HL',IX,IY,SP,PC)
407D	407E	DSKBSP	Disk boot stack pointer beginning location
407F			Unused under Level II
4080	408D	DIVRAM	RAM used with single precision divide
408E	408F	USRADR	USR function address
4090	4092	RNDMUL	Mantissa of multiplicative constant for RND
4093	4095	INPRAM	INP function (93 = "IN" instruction, 94 = port, 95 = Ret)
4096	4098	OUTRAM	OUT function (96=Out,97=port,98=Ret)
4099		KEYBUF	Inkey\$ buffer or flag (last key hit on keyboard)
409A		ERRNBR	Level II Error
409B		LPTPOS	Line printer line position
409C		OUTBFL	Output bit flag: 0=Video, 1=Lp, 80=Cassette
409D		LINLEN	Maximum length of a line on the screen
409E		PRNTZN	Next print zone (reached after a comma as in ?A,B,C)
409F			Unused under Level II
40A0	40A1	STRNGS	Beginning of string area
40A2	40A3	CURLIN	Current line number
40A4	40A5	PGMBGN	Pointer to start of BASIC program
40A6		CRTPOS	Current line position on Video
40A7	40A8	INBUFP	Input buffer pointer
40A9		DATAFL	Data statement flag
40AA	40AC	RNSEED	RND function seed
40AD			Unused under Level II
40AE		DLFLG	Dimension/Let flag from parser
40AF		TYPFLG	Current variable type (8=DBL, 4=SGL, 3=STR, 2=INT)
40B0		TYPFL2	Variable type for FPA2
40B1	40B2	LSTBYT	Address of last usable byte in memory (BASIC)
40B3	40B4	STRPTR	String parameter pointer
40B5	40D2	STPRMS	String param. area. 3 byte sets. 1ST=Length, 2-3=Address
40D3		STRLEN	Length of current string
40D4	40D5	STRADR	Address of current string
40D6	40D7	STRFRE	Next free byte in string area
40D8	40D9	CURTKN	Stores pointer to current token
40D8		PUCBYT	Printusing control byte: Bit2=*,3=+,4=\$,6=Comma
40DA	40DB	LSTDTL	Last data line number read
40DC		FORFLG	Set to 64 on FOR loop. Prevent subscripted variable.
40DE		RDINFL	Read/Input flag: Non-zero=read / Zero = input
40DF	40E0	TRAADR	Transfer address for system
40E1		AUTOFL	Auto flag (Non-zero=ON. Zero after BREAK)
40E2	40E3	AUTOLN	Auto line number
40E4	40E5	AUTINC	Auto increment
40E6	40E7	LLEND	Points to end of previous line or current line
40E8	40E9	SPSAV	Stack pointer save area
40EA	40EB	ERRLIN	Line containing error
40EC	40ED	CURNUM	Current line number
40EE	40EF	PLEND	Pointer to previous line end

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
40F0	40F1	ERRPRC	Address of "ON ERROR"
40F2		ERRFLG	FFH after error. Zero if no error
40F7	40F8	NBIBP	Ptr to next byte to be used with "CONT"
40F9	40FA	SCLERS	Pointer to beginning of scalars
40FB	40FC	ARRAYS	Pointer to beginning of arrays
40FD	40FE	ENDVAR	End location of array variables
40FF	40A0	RESTLN	Used with RESTORE. Keeps current line number for "READ"
40FF	4100	DATPTR	Pointer to delimiter after last DATA Value read
4101	411A	TYPTBL	Variable types for each letter A-Z
411B		TRCFLG	TRON - AF, TROFF - 0
411D	4124	REAL8	Double precision variable
411D	4120	REAL8M	Extended mantissa : Double precision
4121	4124	FPA1	Floating Point Accumulator
4121	4123	FPA1M	Mantissa
4124		FPA1E	Characteristic (exponent)
4125		EXPWRK	Exponent work area
4127	412E	FPA2	Floating Point Accumulator #2
4130		ASCBUF	Numeric work area: converted binary to ASCII number
4152	41A5	DBJPVS	Disk BASIC jump vectors
4152		CVI	CVI: (DBcmd E6H)
4155		FN	FN: (DBcmd BEH)
4158		CVS	CVS: (DBcmd E7H)
415B		DEF	DEF: (DBcmd B0H)
415E		CVD	CVD: (DBcmd E8H)
4161		EOF	EOF: (DBcmd E9H)
4164		LOC	LOC: (DBcmd EAH)
4167		LOF	LOF: (DBcmd EBH)
416A		MKI	MKI\$: (DBcmd ECH)
416D		MKS	MKS\$: (DBcmd EDH)
4170		MKD	MKD\$: (DBcmd EEH)
4173		CMD	CMD: (DBcmd 85H)
4176		TIME	TIME\$: (DBcmd C7H)
4179		OPEN	OPEN: (DBcmd A2H)
417C		FIELD	FIELD: (DBcmd A3H)
417F		GET	GET: (DBcmd A4H)
4182		PUT	PUT: (DBcmd A5H)
4185		CLOSE	CLOSE: (DBcmd A6H)
4188		LOAD	LOAD: (DBcmd A7H)
418B		MERGE	MERGE: (DBcmd A8H)
418E		NAME	NAME: (DBcmd A9H)
4191		KILL	KILL: (DBcmd AAH)
4197		LSET	LSET: (DBcmd ABH)
419A		RSET	RSET: (DBcmd ACH)
419D		INSTR	INSTR: (DBcmd C5H)
41A0		SAVE	SAVE: (DBcmd ADH)
41A3		LINE	LINE: (DBcmd 9CH)
41A6		ERHOOK	Hook to Disk BASIC for long error msgs.
41E6	42E7	IOBUFF	I/O Buffer
4200	42FF	DOSIOB	DOS I/O buffer for sectors from disk
42E8		CON0	Constant: 0
42E9		WRKRAM	Begin BASIC program and work area

<u>Start</u>	<u>End</u>	<u>Label</u>	<u>Description</u>
4300	5FFF	TRSDOS	DOS routines
4300	4307	DIRTRK	Locations of the directory tracks of the different drives
4308		CURDRV	Current drive being used
4309		CDRVBT	Current drive being used with correct bit pattern already calculated and stored at this address.
430A	430B	CURDCB	Address of currently active DCB
430C	430D	CURBUF	Currently active I/O buffer for file reads/writes.
430E		CUROVL	Current overlay in memory
430F		OVLDBG	Overlay/Debug flag
4315	4317	DEBUGV	Debug vector
4318	4347	DOSBUF	DOS Command buffer
4400		WMSTRT	Warmstart
4405		CMDINT	Command Interpreter entry point
4409		POSTER	Post error message entry point
440D		DEBUG	Enter the real-time debugging facility
4410		ACTINT	Activate an interrupt task
4413		TSKOFF	Turn off an interrupt task
4416		TSKCHG	Change state of an interrupt task
4419		DCTTSK	Deactivate an interrupt task
441C		GTSPEC	Get a file specification from buffer
4420		INIT	INIT (DOS file call. P#6-8)
4424		OPEN	OPEN (DOS file call. P#6-9)
4428		CLOSE	CLOSE (DOS file call. P#6-11)
442C		KILL	KILL (DOS file call. P#6-11)
4430		LOAD	Load a machine language format file
4433		RUN	Load and execute machine language file
4436		READ	READ (DOS file call. P#6-9)
4439		WRITE	WRITE a file by sector or Logical record
443C		VERIFY	Write and verify a file write
443F		REWIND	Rewind a file to the beginning
4442		POSN	POSN (DOS file call. P#6-9)
4445		BKSPA	Backspace a file
4448		POSEOF	Position a file to EOF
4467		OUTLIN	Output a line to the CRT
446A		OUTLP	Output a line to the printer
446D		TIME	Move current TIME to 8-byte HL buffer
4470		DATE	Returns DATE into 8-byte HL buffer
4473		DEFEXT	Add default file extension
4476		OPTION	Get optional command flags from buffer
4FFF		LAD4K	Last RAM address in a 4K TRS-80
51FF		ENDOVR	End of DOS overlay area
5200	6FFF	DSKUTL	Disk BASIC/DOS utilities/User memory
7FFF		LAD16K	Last RAM address in a 16K TRS-80
BFFF		LAD32K	Last RAM address in a 32K TRS-80
FFFF		LAD48K	Last RAM address in a 48K TRS-80

APPENDIX B: INTERFACING EXAMPLES

This appendix contains three programs which utilize the routines described in this book to perform their functions.

The first example is a program which inputs a double precision value from the keyboard and displays the number in hexadecimal form after the machine has converted it. The program includes the error recovery routines.

Beginning on page B-4, one will find a program that solves quadratic equations through the use of the math routines.

The final program which begins on page B-8 is a random number generator test which illustrates the use of the ASCII conversion and the random number generator routines.

These examples can be used by either entering and assembling the source statements provided or by entering the hex object code from the assembled output into memory using T-Bug or a similar utility.

```

00100 ;*****
00110 ; DOUBLE PRECISION NUMBERS IN HEXADECIMAL
00120 ; INCLUDES ERROR RECOVERY PRINCIPLES
00130 ;*****
7000 00140 ORG 7000H
7000 310070 00150 BEGIN LD SP,$ ;SET STACK POINTER
7003 CDC901 00160 CALL 01C9H ;CLEAR SCREEN
00170 ;*****
00180 ; RAM INITIALIZATION ROUTINE
00190 ;*****
7006 118040 00200 LD DE,4080H ;RAM INITIALIZATION
7009 21F718 00210 LD HL,18F7H
700C 012700 00220 LD BC,27H
700F EDB0 00230 LDIR
00240 ;*****
00250 ; SETUP ERROR RECOVERY ROUTINE
00260 ;*****
7011 210070 00270 LD HL,BEGIN
7014 22E840 00280 LD (40E8H),HL
7017 21A641 00290 LD HL,41A6H
701A 36C3 00300 LD (HL),0C3H
701C 23 00310 INC HL
701D 110070 00320 LD DE,BEGIN
7020 73 00330 LD (HL),E
7021 23 00340 INC HL
7022 72 00350 LD (HL),D
7023 3EC9 00360 LD A,0C9H ;SET UP A RETURN
7025 32BE41 00370 LD (41BEH),A
7028 32D041 00380 LD (41D0H),A
702B 32C141 00390 LD (41C1H),A
702E 32F240 00400 LD (40F2H),A ;MUST BE <> 0
00410 ;*****
00420 ; PROMPT USER INPUT OF NUMBER
00430 ;*****
7031 219C70 00440 START LD HL,MSG
7034 CD6370 00450 CALL MSGOUT
00460 ;*****
00470 ; READ INPUT RESPONSE
00480 ;*****
7037 213041 00490 LD HL,4130H ;PT TO INPUT BUFFER
703A 0614 00500 LD B,20 ;SET BUF SIZE
703C CD4000 00510 CALL 0040H ;INPUT USER NUMBER
703F AF 00520 XOR A
7040 B0 00530 OR B ;TEST FOR NO INPUT
7041 CA2D40 00540 JP Z,402DH ;BACK TO DOS
00550 ;*****
00560 ; BUFFER MUST END WITH HEX 00
00570 ;*****
7044 EB 00580 EX DE,HL ;INSERT ENDING ZERO
7045 68 00590 LD L,B ;BYTE COUNT TO BC
7046 2600 00600 LD H,0
7048 19 00610 ADD HL,DE ;PT TO <ENTER> BYTE
7049 3600 00620 LD (HL),0 ;PLACE A <0> INTO BUF
704B EB 00630 EX DE,HL ;RESTORE BUF PTR
00640 ;*****
00650 ; CONVERT ASCII NUMBER IN BUFFER TO DOUBLE PRECISION

```

```

00660 ;*****
704C CD650E 00670 CALL 0E65H ;CVRT TO BINARY
704F 211D41 00680 LD HL,411DH ;POINT TO FPA1 EXTENDED
7052 11B070 00690 LD DE,NUMBUF ;POINT TO MY BUFFER
7055 CDD209 00700 CALL 9D2H ;MOVE FPA1 EXTENDED TO BUF
7058 21B070 00710 LD HL,NUMBUF ;POINT TO MY BUFFER
705B CD7870 00720 CALL HEXOUT ;OUTPUT NUMBER IN HEX
705E CD6C70 00730 CALL WRRET ;WRITE AN <ENTER>
7061 18CE 00740 JR START ;RECYCLE
00750 ;*****
00760 ; ;MESSAGE OUTPUT ROUTINE
00770 ;*****
7063 7E 00780 MSGOUT LD A,(HL) ;P/U CHARACTER
7064 B7 00790 OR A ;TEST FOR ZERO
7065 C8 00800 RET Z ;RET IF SO
7066 CD7270 00810 CALL WRBYT ;ELSE OUTPUT IT
7069 23 00820 INC HL ;BUMP POINTER
706A 18F7 00830 JR MSGOUT ;AND LOOP
00840 ;*****
00850 ; MISCELLANEOUS OUTPUT ROUTINES
00860 ;*****
706C 3E0D 00870 WRRET LD A,0DH ;WRITE AN <ENTER>
706E 1802 00880 JR WRBYT
7070 3E20 00890 WRSPA LD A,20H ;WRITE A <SPACE>
7072 D5 00900 WRBYT PUSH DE ;WRITE ANY CHARACTER
7073 CD3300 00910 CALL 33H
7076 D1 00920 POP DE
7077 C9 00930 RET
7078 CD7B70 00940 HEXOUT CALL WRDBL ;CVRT 4 BYTES
707B CD7E70 00950 WRDBL CALL WRHEX ;CVRT 2 BYTES
707E CD7070 00960 WRHEX CALL WRSPA
7081 CD8470 00970 CALL WR2 ;CVRT 1 BYTE
7084 7E 00980 WR2 LD A,(HL) ;P/U BYTE TO CONVERT
7085 CB3F 00990 SRL A ;SHIFT HIGH NYBBLE
7087 CB3F 01000 SRL A ;INTO LOW ORDER POSITION
7089 CB3F 01010 SRL A
708B CB3F 01020 SRL A
708D CD9470 01030 CALL WRDIG ;OUTPUT A HEX DIGIT
7090 7E 01040 LD A,(HL) ;P/U THE BYTE AGAIN
7091 E60F 01050 AND 0FH ;STRIP HIGH NYBBLE
7093 23 01060 INC HL ;PT TO NEXT BUFFER BYTE
7094 C690 01070 WRDIG ADD A,90H ;CVRT A LOW ORDER NYBBLE
7096 27 01080 DAA ;TO ASCII HEX
7097 CE40 01090 ADC A,40H
7099 27 01100 DAA
709A 18D6 01110 JR WRBYT ;OUTPUT THE CHARACTER
01120 ;*****
01130 ; DATA AREA
01140 ;*****
709C 45 01150 MSG DEFM 'ENTER YOUR NUMBER >'
4E 54 45 52 20 59 4F 55
52 20 4E 55 4D 42 45 52
20 3E
70AF 00 01160 NOP
0008 01170 NUMBUF DEFS 8
7000 01180 END BEGIN

```

```

00010 ;*****
00020 ;      SOLUTION OF QUADRATIC EQUATION
00030 ;      VIA MACHINE LANGUAGE INTERFACE
00060 ;*****
7000      00070      ORG      07000H
7000 310070 00080 BEGIN  LD      SP,$          ;SET STACK POINTER
7003 CDC901 00090      CALL    01C9H          ;CLEAR SCREEN
00100 ;*****
00110 ;      RAM INITIALIZATION ROUTINE
00120 ;*****
7006 118040 00130      LD      DE,4080H
7009 21F718 00140      LD      HL,18F7H
700C 012700 00150      LD      BC,27H
700F EDB0   00160      LDIR
00170 ;*****
00180 ;      INIT ERROR RECOVERY
00190 ;*****
7011 210070 00200      LD      HL,BEGIN
7014 22E840 00210      LD      (40E8H),HL
7017 21A641 00220      LD      HL,41A6H
701A 36C3   00230      LD      (HL),0C3H
701C 23     00240      INC     HL
701D 110070 00250      LD      DE,BEGIN
7020 73     00260      LD      (HL),E
7021 23     00270      INC     HL
7022 72     00280      LD      (HL),D
7023 3EC9   00290      LD      A,0C9H          ;SET UP A RETURN
7025 32BE41 00300      LD      (41BEH),A
7028 32D041 00310      LD      (41D0H),A
702B 32C141 00320      LD      (41C1H),A
702E 32F240 00330      LD      (40F2H),A      ;MUST BE <> 0
00340 ;
7031 3E41   00350 START LD      A,'A'          ;INIT FOR 'A' COEFFICIENT
7033 CD1871 00360      CALL    GETNUM
7036 217D71 00370      LD      HL,VALA        ;XFR 'A' TO STORAGE
7039 CDCB09 00380      CALL    09CBH
00390 ;
703C 3E42   00400      LD      A,'B'          ;INIT FOR 'B' COEFFICIENT
703E CD1871 00410      CALL    GETNUM
7041 218171 00420      LD      HL,VALB        ;XFR 'B' TO STORAGE
7044 CDCB09 00430      CALL    09CBH
00440 ;
7047 3E43   00450      LD      A,'C'          ;INIT FOR 'C' COEFFICIENT
7049 CD1871 00460      CALL    GETNUM
704C 218571 00470      LD      HL,VALC        ;XFR 'C' TO STORAGE
704F CDCB09 00480      CALL    09CBH
00490 ;*****
00500 ;      CALCULATE 'B*B'
00510 ;*****
7052 218171 00520      LD      HL,VALB
7055 CDB109 00530      CALL    09B1H          ;'B' TO FPA1
7058 CDBF09 00540      CALL    09BFH          ;'B' TO RFPA FROM FPA1
705B CD4708 00550      CALL    0847H          ;'B' * 'B'
705E CDA409 00560      CALL    09A4H          ;STACK 'B*B'
00570 ;*****
00580 ;      CALCULATE '4AC'

```

```

00590 ;*****
7061 217D71 00600 LD HL,VALA
7064 CDB109 00610 CALL 09B1H ;'A' TO FPA1
7067 218571 00620 LD HL,VALC
706A CDC209 00630 CALL 09C2H ;'C' TO RFPA
706D CD4708 00640 CALL 0847H ;A * C
7070 217871 00650 LD HL,FOUR
7073 CDC209 00660 CALL 09C2H ;'4' TO RFPA
7076 CD4708 00670 CALL 0847H ;4 * A * C
00680 ;*****
00690 ; CALCULATE B*B - 4AC
00700 ;*****
7079 C1 00710 POP BC ;RCVR 'B*B'
707A D1 00720 POP DE
707B CD1307 00730 CALL 0713H ;B*B - 4AC
00740 ;*****
00750 ; TEST DETERMINANT FOR < ZERO
00760 ;*****
707E 217271 00770 LD HL,ZERO
7081 CDC209 00780 CALL 09C2H ;'ZERO' TO RFPA
7084 CD0C0A 00790 CALL 0A0CH ;COMPARE SNGL
7087 2807 00800 JR Z,REAL ;SOLUTION IS IMAGINARY IF
7089 F29070 00810 JP P,REAL ;DETERMINANT IS NEGATIVE
708C CD8209 00820 CALL 0982H ;CHG SIGN OF FPA1
708F 3E 00830 DEFB 3EH ;CONSTRUCT 'LD A,0AFH'
7090 AF 00840 REAL XOR A ;FLAG=0->REAL,FLAG<>0->IMAG
7091 327C71 00850 LD (FLAG),A ;SET REAL/IMGNRY FLAG
7094 CDE713 00860 CALL 13E7H ;TAKE SQUARE ROOT
7097 CDA409 00870 CALL 09A4H ;STACK SQRT(B*B-4AC)
709A CDF970 00880 CALL TWOA ;'2A' TO FPA1
709D C1 00890 POP BC
709E D1 00900 POP DE ;RCVR NUMERATOR
709F CDA208 00910 CALL 08A2H ;SQRT(B*B-4AC)/2A
70A2 218971 00920 LD HL,TEMP
70A5 CDCB09 00930 CALL 09CBH ;& STORE IN TEMP
70A8 CDF970 00940 CALL TWOA ;'2A' TO FPA1
70AB 218171 00950 LD HL,VALB
70AE CDC209 00960 CALL 09C2H ;'B' TO RFPA
70B1 CDA208 00970 CALL 08A2H ;B/2A
70B4 3A7C71 00980 LD A,(FLAG) ;TEST REAL/IMGNRY FLAG
70B7 FE00 00990 CP 0
70B9 201D 01000 JR NZ,IMGNRY
70BB CDA409 01010 CALL 09A4H ;& STACK IT
70BE 218971 01020 LD HL,TEMP ;PT TO TEMP
70C1 CD0B07 01030 CALL 070BH ;& ADD TO B/2A
70C4 CD1271 01040 CALL ANSWER
70C7 218971 01050 LD HL,TEMP
70CA CDB109 01060 CALL 09B1H ;'TEMP' TO FPA1
70CD C1 01070 POP BC
70CE D1 01080 POP DE ;RCVR B/2A
70CF CD1307 01090 CALL 0713H ;B/2A-SQRT(B*B-4AC)/2A
70D2 CD1271 01100 CALL ANSWER
70D5 C33170 01110 JP START ;RECYCLE
01120 ;*****
01130 ; ANSWER IS IMAGINARY - OUTPUT:
01140 ; -B/2A +/- SQRT(ABS(B*B -4AC))I/2A

```

```

01150 ;*****
70D8 CD0971 01160 IMGNRY CALL CVRT ;OUTPUT B/2A
70DB 216771 01170 LD HL,PMMSG
70DE CD3C71 01180 CALL MSGOUT
70E1 218971 01190 LD HL,TEMP
70E4 CDB109 01200 CALL 09B1H ;'TEMP' TO FPA1
70E7 CD7709 01210 CALL 0977H ;TAKE ABS TO ENSURE POS
70EA CD0971 01220 CALL CVRT ;OUTPUT SQRT(B*B-4AC)/2A
70ED 216E71 01230 LD HL,MSG1
70F0 CD3C71 01240 CALL MSGOUT
70F3 CD4571 01250 CALL WRRET
70F6 C33170 01260 JP START ;RECYCLE
01270 ;*****
01280 ; ROUTINE TO MULTIPLY 'A' BY 2.0
01290 ;*****
70F9 217D71 01300 TWA LD HL,VALA
70FC CDB109 01310 CALL 09B1H ;'A' TO FPA1
70FF 217471 01320 LD HL,TWO
7102 CDC209 01330 CALL 09C2H ;'2' TO RFPA
7105 CD4708 01340 CALL 0847H ;2 * A
7108 C9 01350 RET
01360 ;*****
01370 ; CONVERT FPA1 TO ASCII AND OUTPUT TO SCREEN
01380 ;*****
7109 CDBD0F 01390 CVRT CALL 0FBDH ;CVRT FPA1 TO ASCII
710C 213041 01400 LD HL,4130H ;POINT TO ASCBUF
710F C33C71 01410 JP MSGOUT ;OUTPUT & RETURN
7112 CD0971 01420 ANSWER CALL CVRT
7115 C34571 01430 JP WRRET
01440 ;*****
01450 ; ROUTINE TO INPUT COEFFICIENT
01460 ;*****
7118 326371 01470 GETNUM LD (MSGVAR),A ;LOAD COEFF CHAR INTO MSG
711B 215171 01480 LD HL,MSG1
711E CD3C71 01490 CALL MSGOUT ;OUTPUT MSG
7121 213041 01500 LD HL,4130H ;PT TO INPUT BUFFER
7124 0614 01510 LD B,20 ;SET BUF SIZE
7126 CD4000 01520 CALL 0040H ;INPUT USER NUMBER
7129 AF 01530 XOR A
712A B0 01540 OR B ;TEST FOR NO INPUT
712B CA2D40 01550 JP Z,402DH ;BACK TO DOS
01560 ;*****
01570 ; ASCII BUFFER NEEDS ENDING ZERO BYTE
01580 ;*****
712E EB 01590 EX DE,HL ;INSERT ENDING ZERO
712F 68 01600 LD L,B ;BYTE COUNT TO BC
7130 2600 01610 LD H,0
7132 19 01620 ADD HL,DE ;PT TO <ENTER> BYTE
7133 3600 01630 LD (HL),0 ;PLACE A <0> INTO BUF
7135 EB 01640 EX DE,HL ;RESTORE BUF PTR
01650 ;*****
01660 ; CONVERT ASCII INPUT TO BINARY SNGL PREC
01670 ;*****
7136 CD650E 01680 CALL 0E65H ;CVRT TO BINARY
7139 C3B10A 01690 JP 0AB1H ;ENSURE SNGL PREC & RETURN
01700 ;*****

```

```

01710 ; MESSAGE OUTPUT ROUTINE
01720 ;*****
713C 7E 01730 MSGOUT LD A,(HL) ;P/U A CHARACTER
713D B7 01740 OR A
713E C8 01750 RET Z ;RETURN IF CHAR IS ZERO
713F CD4B71 01760 CALL WRBYT ;ELSE OUTPUT IT
7142 23 01770 INC HL ;BUMP POINTER
7143 18F7 01780 JR MSGOUT ;AND LOOP
01790 ;*****
01800 ; MISCELLANEOUS OUTPUT ROUTINES
01810 ;*****
7145 3E0D 01820 WRRET LD A,0DH ;WRITE AN <ENTER>
7147 1802 01830 JR WRBYT
7149 3E20 01840 WRSPA LD A,20H ;WRITE A <SPACE>
714B D5 01850 WRBYT PUSH DE ;WRITE ANY CHARACTER
714C CD3300 01860 CALL 33H
714F D1 01870 POP DE
7150 C9 01880 RET
01890 ;*****
01900 ; BEGINNING OF DATA AREA
01910 ;*****
7151 45 01920 MSG1 DEFM 'ENTER COEFFICIENT '
4E 54 45 52 20 43 4F 45
46 46 49 43 49 45 4E 54
20
7163 58 01930 MSGVAR DEFM 'X >'
20 3E
7166 00 01940 NOP
7167 20 01950 PMMSG DEFM ' +/- '
20 2B 2F 2D 20
716D 00 01960 NOP
716E 20 01970 MSG1 DEFM ' | '
20 49
7171 00 01980 NOP
7172 0000 01990 ZERO DEFW 0 ;FLTG POINT ZERO
7174 0000 02000 TWO DEFW 0 ;FLTG POINT 2.0
7176 0082 02010 DEFW 8200H
7178 0000 02020 FOUR DEFW 0 ;FLTG POINT 4.0
717A 0083 02030 DEFW 8300H
0001 02040 FLAG DEFS 1 ;REAL/IMGNRY FLAG
0004 02050 VALA DEFS 4 ;SPACE FOR COEFFICIENTS
0004 02060 VALB DEFS 4
0004 02070 VALC DEFS 4
0004 02080 TEMP DEFS 4 ;TEMPY STORAGE
7000 02090 END BEGIN
00000 TOTAL ERRORS

```

```

00100 ;*****
00110 ;      RANDOM NUMBER GENERATOR TEST
00120 ;
00130 ;      ILLUSTRATES USE OF ASCII CONVERSIONS
00140 ;      AND RANDOM NUMBER INTERFACING
00150 ;*****
7000      00160      ORG      7000H
7000 310070 00170 BEGIN  LD      SP,$          ;SET STACK POINTER
7003 CDC901 00180      CALL    1C9H          ;CLEAR THE SCREEN
7006 CD5070 00190      CALL    INIT        ;SETUP DIVIDE PROG
00200 ;*****
00210 ;      PROMPT INPUT OF LIMIT
00220 ;*****
7009 217170 00230 START  LD      HL,MSG        ;POINT TO INPUT BUFFER
700C CD5C70 00240      CALL    MSGOUT       ;OUTPUT MESSAGE
700F 213041 00250      LD      HL,4130H     ;POINT TO INPUT BUFFER
7012 0614   00260      LD      B,20        ;SET BUFFER SIZE
7014 CD4000 00270      CALL    40H         ;INPUT LIMIT
7017 AF     00280      XOR      A          ;TEST FOR NO INPUT
7018 B0     00290      OR      B
7019 CA2D40 00300      JP      Z,402DH      ;EXIT (TO 6CCH FOR BASIC)
00310 ;*****
00320 ;      END OF INPUT MUST BE HEX 00
00330 ;*****
701C EB     00340      EX      DE,HL        ;BUFFER ADDRESS TO DE
701D 68     00350      LD      L,B         ;BYTE COUNT TO HL
701E 2600   00360      LD      H,0
7020 19     00370      ADD     HL,DE        ;PT TO <ENTER> BYTE
7021 3600   00380      LD      (HL),0      ;PLACE A ZERO INTO PLACE
7023 EB     00390      EX      DE,HL        ;REPOINT HL TO BUFFER
00400 ;*****
00410 ;      CONVERT ASCII NUMBER TO DOUBLE PRECISION
00420 ;*****
7024 CD650E 00430      CALL    0E65H        ;CVRT TO BINARY
00440 ;*****
00450 ;      EXERCISE 'CSNG' FUNCTION
00460 ;*****
7027 CDB10A 00470      CALL    0AB1H        ;CVRT TO SNGL
00480 ;*****
00490 ;      NOW GENERATE 256 RANDOM NUMBERS
00500 ;*****
702A 0600   00510      LD      B,0          ;INIT COUNTER
702C 218470 00520      LD      HL,NUM       ;
702F CDCB09 00530      CALL    9CBH         ;SAVE FPA1 IN 'NUM'
7032 C5     00540 LOOP  PUSH    BC         ;SAVE COUNTER
7033 218470 00550      LD      HL,NUM       ;LOAD LIMIT
7036 CDB109 00560      CALL    9B1H         ;INTO FPA1 EACH ITERATION
7039 CDEF0A 00570      CALL    0AEFH        ;SET TYPFLG TO 4
00580 ;*****
00590 ;      GEN SNGL PREC RANDOM NUMBER IN FPA1
00600 ;*****
703C CDC914 00610      CALL    14C9H        ;GEN RANDOM NUMBER
00620 ;*****
00630 ;      CONVERT FPA1 TO ASCII
00640 ;*****
703F CDBD0F 00650      CALL    0FBDH        ;CVRT TO ASCII

```

```

00660 ;*****
00670 ; OUTPUT TO CRT DEVICE
00680 ;*****
7042 213041 00690 LD HL,4130H ;POINT TO BUFFER
7045 CD5C70 00700 CALL MSGOUT ;OUTPUT TO CRT
7048 CD6970 00710 CALL WRSPA ;& A <SPACE>
704B C1 00720 POP BC ;RECOVER COUNTER
704C 10E4 00730 DJNZ LOOP ;CYCLE IF MORE
00740 ;*****
00750 ; CYCLE FOR ANOTHER NUMBER
00760 ;*****
704E 18B9 00770 JR START
00780 ;*****
00790 ; RAM INITIALIZATION ROUTINE
00800 ;*****
7050 118040 00810 LD DE,4080H
7053 21F718 00820 LD HL,18F7H
7056 012700 00830 LD BC,27H
7059 EDB0 00840 LDIR
705B C9 00850 RET
00860 ;*****
00870 ; ;MESSAGE OUTPUT ROUTINE
00880 ;*****
705C 7E 00890 MSGOUT LD A,(HL) ;P/U CHARACTER
705D B7 00900 OR A ;TEST FOR ZERO
705E C8 00910 RET Z ;RET IF SO
705F CD6B70 00920 CALL WRBYT ;ELSE OUTPUT IT
7062 23 00930 INC HL ;BUMP POINTER
7063 18F7 00940 JR MSGOUT ;AND LOOP
00950 ;*****
00960 ; MISCELLANEOUS OUTPUT ROUTINES
00970 ;*****
7065 3E0D 00980 WRRET LD A,0DH ;WRITE AN <ENTER>
7067 1802 00990 JR WRBYT
7069 3E20 01000 WRSPA LD A,20H ;WRITE A <SPACE>
706B D5 01010 WRBYT PUSH DE ;WRITE ANY CHARACTER
706C CD3300 01020 CALL 33H
706F D1 01030 POP DE
7070 C9 01040 RET
01050 ;*****
01060 ; DATA AREA
01070 ;*****
7071 45 01080 MSG DEFM 'ENTER UPPER LIMIT'
4E 54 45 52 20 55 50 50
45 52 20 4C 49 4D 49 54

7082 0D 01090 DEFB 0DH
7083 00 01100 NOP
0004 01110 NUM DEFS 4
7000 01120 END BEGIN
00000 TOTAL ERRORS

```


APPENDIX C: BASIC DISASSEMBLER

The listing which follows is a BASIC language, Z-80 disassembler. It can be used to complete the ROM listing provided in Chapter 4 by those readers who own a TRS-80 microcomputer but do not own a machine language disassembler. Entry of the starting and ending addresses is in decimal.

```

1      CLEAR 250
10     DEFINT A-Z
15     HEX$="0123456789ABCDEF"
20     CLS:
      PRINT CHR$(23):
      PRINT STRING$(27,"*")
21     PRINT " DISASSEMBLER BY MISOSYS *":
      PRINT STRING$(27,"*"):
      PRINT"READING DATABASE...."
50     DIM SIZE%(255),ED$(56),ED$(56),OPTBL$(255),
      CODE$(15),ARG$(7),CBTBL$(10),
      DDFD$(38),DDFD$(38),DF(38)
100    FOR I=0 TO 63:
      READ SIZE%(I):
      NEXT I:
      REM LENGTH TABLE
110    DATA 1,3,1,1,1,1,2,1,1,1,1,1,1,1,2,1:
      REM 00 - 0F
120    DATA 2,3,1,1,1,1,2,1,2,1,1,1,1,2,2,1:
      REM 10 - 1F
130    DATA 2,3,3,1,1,1,2,1,2,1,3,1,1,1,2,1:
      REM 20 - 2F
140    DATA 2,3,3,1,1,1,2,1,2,1,3,1,1,1,2,1:
      REM 30 - 3F
150    FOR I=64 TO 191:
      SIZE%(I)=1:
      NEXT:
      REM 40 - BF
160    FOR I=192 TO 255:
      READ SIZE%(I):
      NEXT:
      REM READ REMAINDER
170    DATA 1,1,3,3,3,1,2,1,1,1,3,4,3,3,2,1:
      REM C0 - CF
180    DATA 1,1,3,2,3,1,2,1,1,1,3,2,3,4,2,1:
      REM D0 - DF
190    DATA 1,1,3,1,3,1,2,1,1,1,3,1,3,4,,1:
      REM E0 - EF

```

```

200 DATA 1,1,3,1,3,1,2,1,1,1,3,1,3,4,2,1:
    REM F0 - FF
300 FOR I=1 TO 56:
    READ ED%(I),ED$(I):
    NEXT:
    REM READ ED TABLES
310 DATA 64,"IN B,(C)",65,"OUT (C),B",66,"SBC HL,BC"
320 DATA 67,BC,68,NEG,69,RETN,70,IM 0,71,
    "LD I,A",72,"IN C,(C)"
330 DATA 73,"OUT (C),C",74,"ADC HL,BC",75,
    BC,77,RETI,79,"LD R,A"
340 DATA 80,"IN D,(C)",81,"OUT (C),D",82,
    "SBC HL,DE",83,DE
350 DATA 86,IM 1,87,"LD A,I",88,"IN E,(C)",89,
    "OUT (C),E"
360 DATA 90,"ADC HL,DE",91,DE,94,IM 2,95,
    "LD A,R",96,"IN H,(C)"
370 DATA 97,"OUT (C),H",98,"SBC HL,HL",103,RRD,
    104,"IN L,(C)"
380 DATA 105,"OUT (C),L",106,"ADC HL,HL"
385 DATA 111,RLD,114,"SBC HL,SP"
390 DATA 115,SP,120,"IN A,(C)",121,"OUT (C),A",
    122,"ADC HL,SP"
400 DATA 123,SP,160,LDI,161,CPI,162,INI,163,
    OUTI,168,LDD
410 DATA 169,CPD,170,IND,171,OUTD,176,LDIR,
    177,CPIR,178,INIR
420 DATA 179,OTIR,184,LDDR,185,CPDR,186,INDR,187,OTDR
430 FOR I=0 TO 15:
    READ CODE$(I):
    NEXT
440 DATA "LD B,","LD C,","LD D,","LD E,"
450 DATA "LD H,","LD L,","LD (HL),","LD A,"
460 DATA "ADD A,","ADC A","SUB ","SBC A,"
470 DATA "AND ","XOR ","OR ","CP "
480 FOR I=0 TO 7:
    READ ARG$(I):
    NEXT
490 DATA B,C,D,E,H,L,(HL),A
500 FOR I=0 TO 127:
    READ OPTBL$(I):
    NEXT:
    REM READ PARTIAL OPS
510 DATA NOP,"LD BC,","LD (BC),A",INC BC
520 DATA INC B,DEC B,"LD B,","RLCA
530 DATA "EX AF,AF'", "ADD HL,BC", "LD A,(BC)",DEC BC
540 DATA INC C,DEC C,"LD C,","RRCA
550 DATA "DJNZ,","LD DE,","LD (DE),A",INC DE
560 DATA INC D,DEC D,"LD D,","RLA
570 DATA JR ,"ADD HL,DE", "LD A,(DE)",DEC DE
580 DATA INC E,DEC E,"LD E,","RRA

```

```

590 DATA "JR NZ,","LD HL,","",INC HL
600 DATA INC H,DEC H,"LD H","",DAA
610 DATA "JR Z,","ADD HL,HL","LD HL,","DEC HL
620 DATA INC L,DEC L,"LD L","",CPL
630 DATA "JR NC,","LD SP,","",INC SP
640 DATA INC (HL),DEC (HL),"LD (HL),","SCF
650 DATA "JR C,","ADD HL,SP","LD A,","DEC SP
660 DATA INC A,DEC A,"LD A","",CCF
670 DATA RET NZ,POP BC,"JP NZ,","JP "
680 DATA "CALL NZ,","PUSH BC,"ADD A,","RST 00H
690 DATA RET Z,RET,"JP Z,",""
700 DATA "CALL Z,","CALL ","ADC A,","RST 08H
710 DATA RET NC,POP DE,"JP NC,","OUT N,"
720 DATA "CALL NC,","PUSH DE,SUB ,RST 10H
730 DATA RET C,EXX,"JP C,","IN A,"
740 DATA "CALL C,","","SBC A,","RST 18H
750 DATA RET PO,POP HL,"JP PO,","EX (SP),HL"
760 DATA "CALL PO,","PUSH HL,AND ,RST 20H
770 DATA RET PE,JP (HL),"JP PE,","EX DE,HL"
780 DATA "CALL PE,","","XOR ,RST 28H
790 DATA RET P,POP AF,"JP P,","DI
800 DATA "CALL P,","PUSH AF,OR ,RST 30H
810 DATA RET M,"LD SP,HL","JP M,","EI
820 DATA "CALL M,","","CP ,RST 38H
830 FOR I=0 TO 10:
    READ CBTBL$(I):
    NEXT:
    REM READ CB OP TABLE
840 DATA RLC,RRC,RL,RR,SLA,SRA,XXX,SRL,BIT,"RES","SET"
850 FORI=0 TO 38:
    READ DDFD%(I),DDFD$(I):
    NEXT:
    REM READ DD & FD TABLES
860 DATA 9,"ADD X,BC",25,"ADD X,DE",33,"X"
870 DATA 34,X,35,"INC X",41,"ADD X,X",42,X
880 DATA 43,DEC X,52,INC (X+),53,DEC (X+),54,"LD (X+),"
890 DATA 57,"ADD X,SP",70,"LD B,(X+)",78,"LD C,(X+)"
900 DATA 86,"LD D,(X+)",94,"LD E,(X+)",102,"LD H,(X+)"
910 DATA 110,"LD L,(X+)",112,"LD (X+),B",113,"LD (X+),C"
920 DATA 114,"LD (X+),D",115,"LD (X+),E",116,"LD (X+),H"
930 DATA 117,"LD (X+),L",119,"LD (X+),A",126,
    "LD A,(X+)",134,"ADD A,(X+)"
940 DATA 142,"ADC A,(X+)",150,"SUB (X+)",158,
    "SBC A,(X+)"
950 DATA 166,"AND (X+)",174,"XOR (X+)",182,"OR (X+)"
960 DATA 190,"CP (X+)",225,POP X,227,"EX (SP),X
970 DATA 229,PUSH X,233,JP (X),249,"LD SP,X"
980 FOR I=0 TO 38:
    READ DF(I):
    NEXT:
    REM READ ED/FD "LENGTH" TABLE

```

```

990      DATA 1,1,3,3,1,1,3,1,2,2,2,1,2,2,2,2,2,2,
           2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1
2000     INPUT "ENTER STARTING ADDRESS";LO
2001     INPUT "ENTER ENDING ADDRESS";HI
2005     PC=LO:
        CLS
2010     BYTE=PEEK(PC):
        GOSUB 12000:
        D=0:
        REM INITIALIZE DISPLACEMENT
2020     N=SIZE%(BYTE):
        ON N GOTO 2030,2040,2050,2060
2030     GOTO2100:
        REM BRANCH FOR STATEMENTS OF LENGTH 1
2040     GOTO2500:
        REM BRANCH FOR STATEMENTS OF LENGTH 2
2050     GOTO2600:
        REM BRANCH FOR STATEMENTS OF LENGTH 3
2060     GOTO2900:
        REM BRANCH FOR LENGTH CODE OF 4
2100     GOSUB 11000
2120     IF BYTE>63 AND BYTE<192 THEN 2200
2130     X=BYTE:
        IF X>191 THEN X=X-128
2140     PRINT TAB(16) OPTBL$(X);
2150     GOSUB 10000
2160     GOTO5000
2200     X=INT((BYTE-64)/8)
2210     IF BYTE <> 118 THEN 2230
2220     PRINT TAB(16) "HALT";:
        GOSUB10000:
        GOTO5000
2230     PRINT TAB(16) CODE$(X);
2240     R=BYTE AND 7
2250     PRINT ARG$(R);
2260     GOSUB 10000:
        GOTO5000
2500     IF BYTE=211 THEN 2560
2510     GOSUB 11000
2520     X=BYTE:
        IF X>191 X=X-128
2530     PRINT TAB(16) OPTBL$(X);
2540     N2=PEEK(PC+1):
        GOSUB12030:
        PRINT"H";
2545     N1=BYTE AND 199:
        IF N1=0 THEN D=N2
2550     GOSUB10000:
        GOTO5000
2560     GOSUB11000:
        PRINT TAB(16)"UT ";

```

```

2570  Y=PC:
      X=N:
      N=1:
      GOSUB11000:
      N=X:
      PC=Y:
      PRINT"H,A";
2580  GOSUB 10000:
      GOTO5000
2600  GOSUB 11000
2610  IF BYTE=34 THEN 2710
2620  IF BYTE=50 THEN 2810
2630  X=BYTE:
      IF X>191 X=X-128
2635  PRINT TAB(16) OPTBL$(X);:
      IFX=42 OR X=58 THEN PRINT"(";
2640  Z=N:
      Y=PC:
      N=2:
      PC=PC+1:
      GOSUB11090:
      N=Z:
      PC=Y:
      IF X=42 OR X=58 THEN PRINT "H)";
      ELSE PRINT"H";
2660  GOSUB10000:
      GOTO5000
2710  PRINT TAB(16)"LD (";
2720  X=N:
      Y=PC:
      N=2:
      PC=PC+1:
      GOSUB11090:
      PC=Y:N=X
2730  PRINT "H),HL";:
      GOSUB10000:
      GOTO5000
2810  PRINT TAB(16)"LD (";
2820  X=N:
      Y=PC:
      PC=PC+1:
      N=2:
      GOSUB11090:
      PC=Y:
      N=X
2830  PRINT "H),A";:
      GOSUB10000:
      GOTO5000
2890  GOSUB11000:
      GOSUB10000:
      GOTO5000

```

```

2900 IF BYTE<>203 THEN 3000
2910 NB=PEEK(PC+1):
      REM DISASSEMBLE 'CB' INSTRUCTIONS
2920 X=INT(NB/8):
      IF X>7 THEN X=INT(X/8)+7
2930 N=2:
      REM RESET LENGTH TO 2
2940 GOSUB11000:
      REM PRINT INSTRUCTION IN HEX
2950 PRINT TAB(16) CBTBL$(X);" ";;
      REM RECOVER OP
2970 IF X<8 THEN 2990
      ELSE X=INT((NB-64)/8)
2975 IF X<8 THEN 2980
2976 X=X-8:
      GOTO2975
2980 PRINTX;" ";;
      REM PRINT THE BIT
2990 Y=NB AND 7:
      PRINT ARG$(Y);:
      GOSUB 10000:
      GOTO5000
3000 REM ***** THIS SECTION DECODES 'ED' INSTRUCTIONS
3010 IF BYTE<>237 THEN 3200:
      REM 237(10)=ED(16)
3020 NB=PEEK(PC+1):
      FOR I=1 TO 56:
          IF NB=ED%(I) THEN3040
          ELSE NEXT
3030 N=1:
      GOSUB11000:
      GOSUB 10000:
      GOTO5000:
      REM INVALID CODE, PRINT ASCII
3040 NX=I:
      IF NB=67 OR NB=83 OR NB=115 THEN 3070
3050 IF NB=75 OR NB=91 OR NB=123 THEN 3100
3060 N=2:
      GOSUB11000:
      PRINT TAB(16) ED$(NX);:
      GOSUB 10000:
      GOTO5000
3070 N=4:
      GOSUB11000:
      PRINT TAB(16)"LD (";
3080 Y=PC:
      PC=PC+2:
      N=2:
      GOSUB11000:
      N=4:
      PC=Y:

```

```

PRINT"H),";ED$(NX);
3090 GOSUB10000:
GOTO5000
3100 N=4:
GOSUB11000:
PRINT TAB(16)"LD ";ED$(NX);",(";
3110 Y=PC:
PC=PC+2:
N=2:
GOSUB11000:
N=4:
PC=Y:
PRINT"H)";
3120 GOSUB10000:
GOTO5000
3200 REM ***** THIS SECTION DECODES 'Dd' & 'FD'
INSTRUCTIONS *****
3210 NB=PEEK(PC+1):
IF NB=203 THEN 3400:
REM CHECK FOR BYTE 2=CB
3220 FOR X=0 TO 38:
IF NB=DDFD%(X) THEN 3240
ELSE NEXT
3230 N=1:
GOSUB11000:
GOSUB10000:
GOTO5000:
REM INVALID OP CODE
3240 P$=DDFD$(X):
Q$="":
FOR I=1 TO LEN(P$):
IF MID$(P$,I,1)<>"X" OR NB=174 OR NB=227 THEN
Q$=Q$+MID$(P$,I,1)
ELSE IF BYTE=221 THEN Q$=Q$+"IX"
ELSE Q$=Q$+"IY"
3250 NEXT:
REM Q$ NOW CONTAINS PARTIAL INST FOR IX/IY
3260 ON DF(X) GOTO 3270,3280,3340
3270 N=2:
GOSUB11000:
PRINT TAB(16) Q$;:
GOSUB10000:
GOTO5000
3280 N=3:
IF NB=54 THEN N=4
3290 GOSUB11000:
P$=Q$:
Q$="":
FOR I=1 TO LEN(P$):
IF MID$(P$,I,1)<>"+" THEN Q$=Q$+MID$(P$,I,1)
ELSE 3310

```

```

3300  NEXT:
      STOP
3310  PRINT TAB(16) Q$;"+";:
      N2=PEEK(PC+2):
      D=N2:
      GOSUB12030:
      PRINT "H";:
      PRINT RIGHT$(P$,LEN(P$)-I);:
      IF NB<>54 THEN 3330
3320  N2=PEEK(PC+3):
      GOSUB12030:
      PRINT"H";
3330  GOSUB10000:
      GOTO5000
3340  IF NB=33 THEN 3350
3342  IF NB=34 THEN 3360
3344  IF NB=42 THEN 3370
      ELSE 3030
3350  N=4:
      GOSUB11000:
      PRINT TAB(16)"LD ";Q$;
3355  PRINT ", ";:
      Y=PC:
      N=2:
      PC=PC+2:
      GOSUB11000:
      PRINT"H";:
      PC=Y:
      N=4:
      GOSUB10000:
      GOTO5000
3360  N=4:
      GOSUB11000:
      PRINT TAB(16) "LD (";:
      Y=PC:
      PC=PC+2:
      N=2:
      GOSUB11000:
      PRINT"H), ";Q$;
3365  PC=Y:
      N=4:
      GOSUB10000:
      GOTO5000
3370  N=4:
      GOSUB11000:
      PRINT TAB(16) "LD ";Q$;" , (";:
      Y=PC:
      PC=PC+2:
      N=2:
      GOSUB11000:
      PRINT"H) ";:

```

```

N=4:
PC=Y:
GOSUB10000:
GOTO5000
3400 N=4:
NB=PEEK(PC+3):
BIT=INT((NB AND 56)/8):
X=INT((NB AND 192)/64):
IF X=0 THEN X=BIT
ELSE X=X+7:
REM X NOW INDEXES CBTBL$
3410 GOSUB 11000:
PRINT TAB(16) CBTBL$(X); " ";;
IF X>7 PRINT BIT; ", ";
3420 IF BYTE = 221 PRINT "(IX+";
ELSE PRINT "IY+";
3430 N2=PEEK(PC+2):
D=N2:
GOSUB12030:
PRINT"H) ";;
GOSUB10000:
GOTO5000
5000 IF D=0 THEN 5005
ELSE GOSUB 13000
5005 L=L+1:
IF L<16 THEN 5010
ELSE INPUT " ...WAITING";Z$:
CLS
5006 LO=LO+L:
L=0
5010 PRINT " ":
PC=PC+N:
IF PC<HI+1 THEN 5020
ELSE PRINT LO,"LINES OUTPUT":
END
5020 IF BYTE<>207 THEN 2010
5030 BYTE=PEEK(PC):
D=0:
N=1:
GOSUB12000:
GOSUB11000:
PRINT TAB(16) "DEFB ";
5040 IF BYTE<32 OR BYTE >127 THEN 5050
ELSE PRINT "'";CHR$(BYTE); "'";:
GOSUB10000:
GOTO5005
5050 N2=BYTE:
GOSUB 12030:
PRINT"H) ";;
GOSUB10000:
GOTO5005

```

```

10000  FOR I=0 TO N-1
10020      IF Q>32 AND Q<128 THEN PRINT TAB(32+I)CHR$(Q);
          ELSE PRINT TAB(32+I) ". ";
10030  NEXT I
10040  RETURN
10090  Q=PEEK(PC+I)
11000  FOR I=0 TO N-1:
          GOSUB11010:
          NEXTI:
          RETURN
11010  Q=PEEK(PC+I)
11020  J=INT(Q/16)
11030  K=Q-J*16
11040  J$=MID$(HEX$,J+1,1)
11050  K$=MID$(HEX$,K+1,1)
11060  PRINT J$;K$;:
          RETURN
11090  FOR I=1 TO 2-N STEP -1:
          GOSUB 11010:
          NEXTI:
          RETURN
12000  N2=PC:
          K=VARPTR(N2):
          N1=PEEK(K+1):
          N2=N2 AND 255
12010  J=INT(N1/16):
          K=N1-J*16
12020  Q$=MID$(HEX$,J+1,1)+MID$(HEX$,K+1,1):
          PRINT Q$;
12030  J=INT(N2/16):
          K=N2-J*16
12040  Q$=MID$(HEX$,J+1,1)+MID$(HEX$,K+1,1):
          PRINT Q$;
12050  PRINT TAB(7):
          RETURN
13000  REM ***** ROUTINE TO COMPUTE RELATIVE DISPLACEMENTS
13010  D1=D AND 128:
          IF D1<>0 THEN POKE VARPTR(D)+1,255
13020  D=D+2
13030  Y=PC:
          PC=PC+D:
          PRINT TAB(40) " ";:
          GOSUB 12000:
          PRINT "H";:
          PC=Y:
          RETURN

```

\$3.00 OFF!

VOLUME TWO!

THE BOOK

Insiders Software Consultants, Inc.
P.O. Box 2441, Dept. SUM 1
Springfield, VA 22152

*TRS-80 is a trademark of
Tandy Corp.

☐ Please send me Volume II of THE BOOK
at \$14.95 plus \$1.50 for postage.

NAME: _____

ADDRESS: _____

CITY, STATE: _____ ZIP: _____

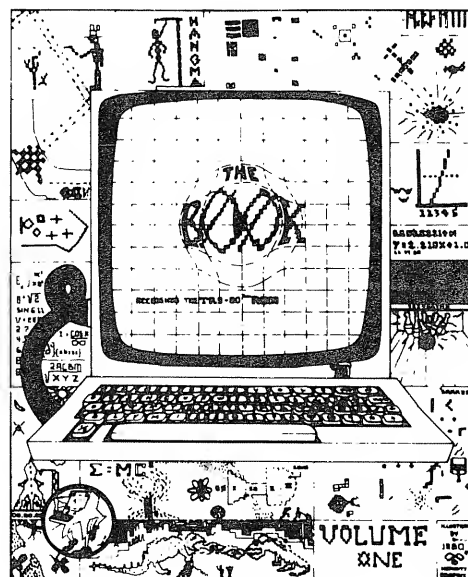
☐ Check payable to Insiders Software Consultants, Inc.

☐ MASTER CHARGE MC Bank Code.

☐ VISA Exp. Date Card Number

Signature: _____

expires december 31, 1981



\$14.95



INSIDERS SOFTWARE CONSULTANTS
P.O. BOX 2441,
SPRINGFIELD, VA 22152